НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ *Сергій СТІРЕНКО*
(підпис)
“___”_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерні системи та мережі»

спеціальності 123 «Комп’ютерна інженерія»

на тему: « IoT система з використанням акселерометра для SmartCity»

Виконав:
студент _IV_курсу, групи__IO-62__
Копійка Антон Олександрович _____
(підпис)

Керівник:
Доцент, кандидат технічних наук
Ткаченко Валентина Василівна _____
(підпис)

Консультант з нормоконтролю:
Доцент, доктор технічних наук
Сімоненко Валерій Павлович _____
(підпис)

Рецензент

_____
(підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМ. ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ *Сергій СТІРЕНКО*
(підпис)
"___"_____20__ р.

ЗАВДАННЯ

на дипломний проєкт студенту

Копійці Анотону Олександровичу

1. Тема проєкту «*IoT система з використанням акселерометра для SmartCity»* керівник проєкту Ткаченко Валентина Василівна, доцент, кандидат технічних наук, затверджені наказом по університету від «__07__» _травня_ 2020р. №__1081-с__

2. Термін подання студентом проєкту _____2020р.

3. Вихідні дані до проєкту технічне завдання, теоретичні дані.

4. Зміст пояснювальної записки: опис предметної області, дослідження систем контролю якості дорожнього покриття, програма, що виконує аналіз якості дорожнього покриття.

5. Консультанти розділів проєкту

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Сімоненко В.П.		

6. Дата видачі завдання 01.09 .2019 року

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів дипломного проєкту	Строк виконання етапів проєкту	Примітки
1.	Затвердження теми роботи	01.08.2019	
2.	Вивчення та аналіз завдання	15.09.2019-15.03.2020	
3.	Розробка архітектури та загальної структури систем	15.03.2020-25.03.2020	
4.	Розробка структур окремих підсистем	25.03.2020-05.04.2020	
5.	Програмна реалізація системи	05.04.2020-15.04.2020	
6.	Оформлення пояснювальної	15.04.2020-20.05.2020	
7.	Передзахист	26.05.2020	
8.	Захист	25.06.2020	

Студент Антон КОПІЙКА _____

(підпис)

Керівник Валентина ТКАЧЕНКО _____

(підпис)

Анотація

В бакалаврській дипломній роботі розроблено систему пошуку дефектів дорожнього покриття з використанням акселерометра, як частину системи IoT.

Дана система може бути використана для інтегрування в крупні IoT системи на кшталт SmartCity, вмонтовані в автомобілі. Продукт був створений за допомогою програмного забезпечення Keil uVision та STM32 CubeMX, на мові C. Програма для відображення результатів роботи реалізована на мові Python.

Annotation

In the bachelor's thesis, a system for finding defects in the road surface using an accelerometer as part of the IoT system was developed.

This system can be used to integrate into large IoT systems such as SmartCity, being installed in the car. The product was created using Keil uVision software and STM32 CubeMX, in C programing language. The program for displaying the results of work is implemented in Python.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ІАЛЦ.467100.001 ВП	Відомість проекту	1	
3	А4	ІАЛЦ.467100.002 ТЗ	Технічне завдання	4	
4	А4	ІАЛЦ.467100.003 ПЗ	Пояснювальна записка	68	
5	А4	ІАЛЦ.467100.004 Д1	Принципова схема апаратного забезпечення системи	1	
6	А4	ІАЛЦ.467100.005 Д2	Структурна схема пристрою	1	
7	А4	ІАЛЦ.467100.006 Д3	Функціональна схема методу визначення ям на дорогах	1	
8	А4	ІАЛЦ.467100.007 Д4	Текст програми	3	

Зм.		№ документа	Підп.	Дата	ІАЛЦ.467100.001 ВП		
Розробив		Копійка А.О.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Ткаченко В,В.				Т	
Н.контр.		Сімоненко В. П.				НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ гр. ІО-62	
Затв.		Стіренко С. Г.					

ТЕХНІЧНЕ ЗАВДАННЯ

до дипломної роботи
освітньо-кваліфікаційного рівня бакалавр

на тему: « IoT система з використанням акселерометра для SmartCity»

Київ – 2020 року

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку курсу «Інженерія програмного забезпечення» та курсу «Захист інформації у комп’ютерних системах». Область застосування: практичне використання людьми під час управління автомобілем, для пошуку дефектів на дорозі.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», затверджене кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проекту є розробка системи пошуку ям на дорогах завдяки акселерометру.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література з теорії і практики програмування, бакалаврські роботи інших студентів, публікації в Інтернеті з даних питань.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробляємого продукту

- Система може оброблювати дані лише з одного датчика.
- Система не повинна оброблювати данні, якщо положення пристрою не природнє.

5.2. Вимоги до програмного забезпечення

- Операційна система MS Windows 7 та новіші версії.
- Наявність на комп’ютері Python не нижче версії 3.7.

5.3. Вимоги до апаратної частини

- Комп’ютер на базі процесора Intel або AMD мінімальної потужності.
- Оперативної пам’яті не менше 512 Мбайт.
- Вільне місце на жорсткому диску не менше 100 Мбайт.

6. ЕТАПИ РОЗРОБКИ

	Дата
Вивчення літератури	30.03.2020
Складання і узгодження технічного завдання	01.04.2020
Створення модулів системи, що розробляється	10.04.2020
Тестування окремих модулів системи	20.04.2020
Допрацювання, налагодження і виправлення помилок	04.05.2020
Оформлення документації дипломної роботи	17.05.2020

ПОЯСНЮВАЛЬНА ЗАПИСКА
до дипломного проєкту
на тему: «IoT система з використанням акселерометра для SmartCity»

Київ – 2020 року

ЗМІСТ

ВСТУП

Щодня всі, без виключення країни світу несуть колосальні збитки в зв'язку з пошкодженнями дорожньої смуги. По-перше, витрачаються мільйони на ремонти та «залатування» пошкоджених ділянок доріг. По-друге, рух по нерівній поверхні автотранспорту збільшує кількість затраченого палива та частіше призводить до потреби в технічному обслуговуванні. Не варто також забувати і про те, що ями на дорогах дуже часто стають однією з причин ДТП і призводять до травм, а іноді навіть до летальних випадків. Ну і що найменше погана якість дорожньої смуги дратує водіїв та створює певний дискомфорт під час руху.

Підтримання доріг в гарному стані є досить складною задачею. Дуже багато факторів впливають на пошкодження. Серед них як природні(кліматичні зміни, опади, деформація ґрунтів, температурні перепади) так і техногенного характеру(знос, перевищення дозволеного тоннажу машин, структурне руйнування шарів, недотримання технічних вимог при прокладанні дороги, тощо). Бюджет дорожньо-ремонтних робіт напряму залежить від швидкості їх реакції на пошкодження. [1] Також не варто забувати, що система, яка б відображала наявність ям на дорогах змогла б допомогти водіям, для побудови кращого маршруту.

Забезпечення контролю якості доріг за допомогою статичних датчиків або людського моніторингу майже неможливе. В зв'язку з надзвичайно великою протяжністю доріг, такі системи потребують коштів, які могли б бути направлені на проведення ремонтних робіт.

З кожним роком все більше набирає популярність концепція Інтернету речей. Сутність даного принципу полягає в обміні даними між датчиками та предметами побуту для людської безпеки або спрощення життєдіяльності. Особливу популярність набирає концепція «SmartCity», в якій елементи інфраструктури спілкуються між собою. Як результат зроблено припущення,

що можливість контролювати якість дорожнього покриття, як елемент такої системи було б надзвичайно корисним доповненням.

З огляду на все вище сказане, можна з впевненістю сказати, що рішення даної проблеми полягає в можливості автотранспортних засобів самим фіксувати проблемні ділянки під час руху. Для реалізації такої системи необхідно лише отримати значення лінійного прискорення, місця розташування і можливість передати ці данні для їх відображення в зручному вигляді.

В даній роботі розглядається задача фіксування дефектів дорожньої смуги за допомогою сучасних датчиків лінійного прискорення для моніторингу якості ділянок дороги.

Було поставлено завдання розглянути алгоритми оцінки якості дорожнього покриття та маркування ділянок доріг за якістю, реалізувати програму, що дозволяє в режимі реального часу виявляти дефекти на поверхні дороги і особливим чином позначати такі ділянки дороги на карті.

Задача полягає в знаходженні певних спільних характеристик, які притаманні всім проблемним ділянкам, та відображання ділянок яким ці характеристики притаманні на карті.

РОЗДІЛ 1
ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Огляд методів збору інформації для визначення якості дорожнього покриття

Задля того, щоб обрати оптимальний спосіб виявлення проблемних ділянок дороги, спочатку необхідно розглянути вже існуючі аналоги систем, які також займалися цим питанням, а саме, якими даними та на які характеристики вони опирались для прийняття рішення стосовно якості покриття.

1.1.1 Метод відеопаспортизації доріг

Пересувні дорожні лабораторії

Найтехнологічнішим способом визначення якості доріг є використання дорожніх лабораторій – це пересувний центр збору даних які пов’язані з дорогою та узбіччям. На таких автомобілях, наприклад як [2] в залежності від поставлених цілей розміщено велику кількість датчиків:

- декілька відеокамер;
- приймач GPS;
- тривимірний лазерний сканер;
- система аналізу ґрунту;
- блок позиціонування.

Такі системи мають широкий спектр аналізу починаючи аналізу. Завдяки такій лабораторії можна отримати:

- аналіз поперечного перерізу шарів асфальту;
- розрахунок осьового відхилення;
- знаходження колій, тріщин та ям;
- перевірку якості розмітки;
- виміри рівності дороги.

Основна частина аналізу відбувається неавтономно по завершенню поїздки. Весь відзнятий відеоматеріал переглядається спеціалістами та на ньому вручну фіксуються всі ями та пошкодження. Даний процес є дуже ресурсо- та часозатратним і ніяк не відповідає можливості використання звичайними водіями без додаткових навичок, тому він абсолютно не відповідає поставленим для роботи задачам.

Автоматизований метод фіксації дорожніх пошкоджень на відео

Іншим варіантом визначення ям та тріщин на дорогах є знаходження їх за допомогою даних у форматі відеозапису представлений в [3]. В основі пошуку вибоїн і заплаток в даному способі лежить використання методу активних контурів [4]. Суть методу полягає в виділенні контуру об’єкта при даній одній точці, яка точно належить об’єкту. Як ініціатор для визначення пошкодження розглядається дисперсія інтенсивності в його внутрішній області [3]. Також, як варіант, розглядається можливість використання штучного інтелекту для пошуку [5].

У даного методу є декілька критичних недоліків, а саме:

- вже відремонтовані ділянки, які відрізняються кольором також фіксуються, як ями;
- розміщення відеокамери має бути ідентичним для всіх автомобілів;
- вихідне відео дає високу похибку, через проблеми з перспективним спотворенням зйомки (об’єкт може бути не помітний повністю, або перекритий тінню).

Опираючись на представлені доводи, цей спосіб також не повністю зможе виконати задачу і має високий рівень похибки.

1.1.2 Мобільний додаток для знаходження ям на дорогах

Наступний варіант вирішення поставленої задачі – мобільний сервіс, для оцінки дорожнього покриття. Вітчизняна компанія створила мобільний додаток для Android або iOS. Проект постійно обновлює данні про якість доріг в Україні і відображає їх за допомогою кольорів на карті, відображаючи погані ділянки червоним, а гарні –зеленим.

Для того щоб долучитися до цієї програми достатньо лише встановити додаток на свій мобільний телефон, зареєструватися, дозволити доступ до GPS і вбудованого в кожен телефон акселерометра та розмістити телефон на торпеді свого авто. Всі сучасні смартфони можуть заміряти силу «струшування» телефону за допомогою акселерометра. Це дозволяє без будь-яких затрат перетворити машину в рухому лабораторію. Зібрана інформація обробляється за допомогою математичних алгоритмів. GPS датчик знаходить поточне розташування і відображає якість на карті. [6,7]

Даний спосіб, мабуть, найекономніший, так як не потребує додаткових затрат, та доступний для всіх власників авто, але такий варіант збору даних досить швидко розрядить ваш мобільний телефон, тому що довге використання GPS датчика дуже ресурсозатратне. Також варто зауважити, що при зрушуванні телефону будуть отримані неправдиві показники, тому під час поїздки не можна використовувати свій телефон, наприклад, як навігатор чи для інших цілей.

1.1.3 Системи які вбудовані в автомобілі

На те, що обрана тема має високу цінність вказує той факт, що великі автомобільні концерни, такі як Ford, Jaguar та Volvo створюють свої власні системи детекції проблемних ділянок доріг. Всі три компанії пропонують схожий варіант системи перевірки якості дороги. За їх словами, нова система зможе знаходити ями, вибоїни та каналізаційні люки, використовуючи сенсори, розташовані під машиною. Ці датчики будуть повідомляти

електроніку машини і в разі необхідності, зменшувати швидкість, або змінювати конфігурацію підвіски для пом’якшення руху машини по нерівній дорозі.

Також данні про знайдені недоліки дорожнього покриття будуть надіслані в хмарні сервіси разом з координатами цього місця. Тоді, якщо інша машина буде наближатися до небезпечної зони, вона буде завчасно повідомлена про проблемну ділянку, та її місцезнаходження. Дана система підтримує парадигму Інтернету речей, в якій машини, за допомогою спільного хмарного сховища зможуть отримати дані, зняті іншими машинами на цій ділянці.

На даному етапі машини можуть перевіряти стан дороги лише на близькій відстані від авто, але в планах розробників встановлення стереоскопічної цифрової камери, котра зможе знаходити ями на достатній відстані для того, щоб встигнути їх оминути, зменшити швидкість, або навіть зупинити машину. [8,9,10]

1.2 Огляд методів обробки інформації для визначення якості дорожнього покриття

В відкритих джерелах розглянуто декілька методів знаходження проблемних ділянок дороги з використанням акселерометрів. Більшість з них представлені в закордонних статтях і використовуються на практиці в аналогічних проектах. Необхідно розглянути всі існуючі варіанти для визначення оптимального саме для нашої системи.

1.2.1 Алгоритми пошуку ям на дорогах

Оскільки дана сфера не до кінця реалізована на практиці і розробляється всесвітньо відомими брендами, то інформації про алгоритми розробки досить мало. Натомість знайдено декілька іноземних статей на дану тему, в яких пояснюються їх методи роботи.

Z –THRESH

Найпростішим та найпоширенішим алгоритмом пошуку ям на дорогах є Z –THRESH. Його суть полягає в відслідковуванні показників акселерометра по вертикальній осі Z (рис. 1.1). В статті [16] він був протестований на зібраному наборі даних і встановлює пороговий показник допустимої амплітуди коливань значень, перевищення якого свідчить про знаходження ями. Алгоритм передбачає, що положення осей акселерометра відомо і не потребує додаткового переорієнтування. В представленій доповіді акселерометр знаходиться в положенні паралельному дорозі. Це зменшує складність алгоритму і апаратні затрати.

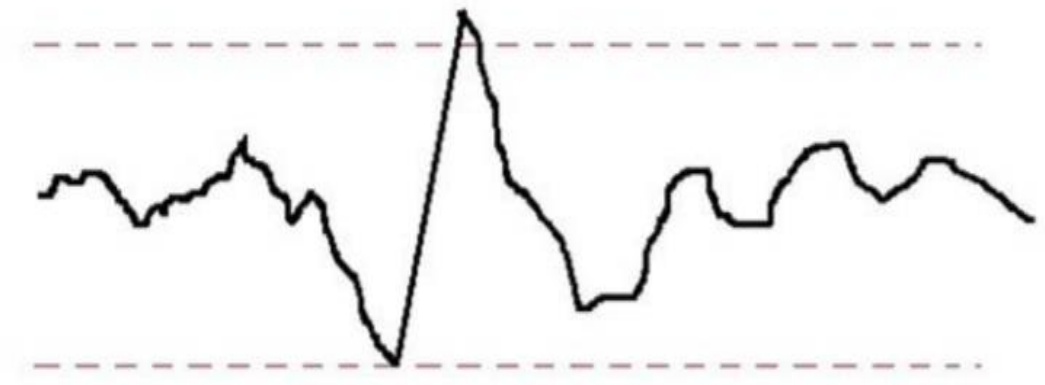


Рис. 1.1. Пошук ям за допомогою алгоритму Z –THRESH [16]

Z – DIFF

Даний метод також полягає на відслідковуванні показників з осі Z, але він є більш складнішим за Z –THRESH. Він потребує аналогічних даних, що і попередній алгоритм, але на відміну від нього, цей алгоритм відслідковує не перевищення показника лінійного прискорення відносно стану спокою. Головним критерієм для Z-DIFF є різка зміна лінійного прискорення між двома послідовними замірами (рис. 1.2). Цей алгоритм визначає швидкі зміни вертикального прискорення, і класифікує їх як можливі ями.

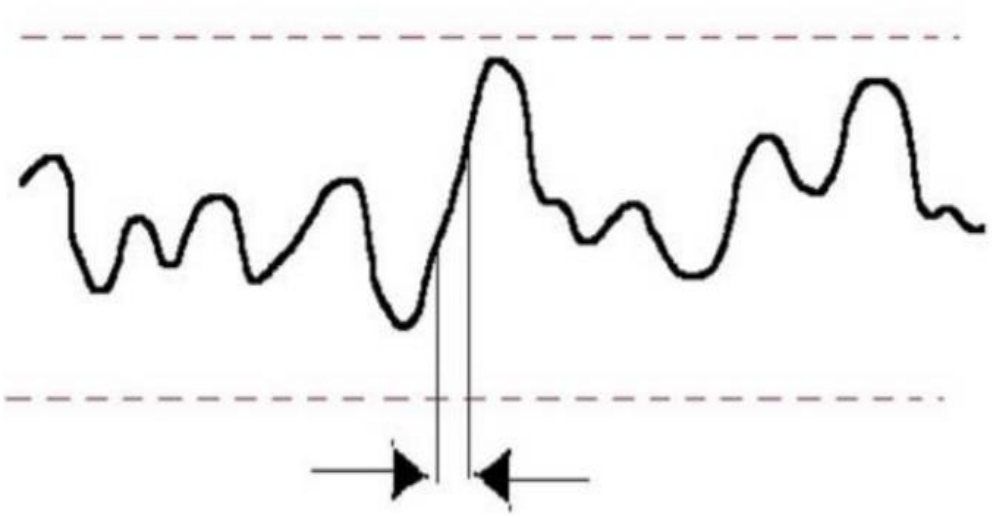


Рис. 1.2. Пошук ям за допомогою алгоритму Z –DIFF [16]

G-ZERO

Даний метод був виявлений завдяки візуальному спостереженню за показами трьохосьового акселерометра протягом дослідження двох попередніх алгоритмів. Назва алгоритму точно ілюструє його суть. Згідно спостереженням, під час руху в певний момент часу показники лінійного прискорення на всіх осях рівні нулю (рис1.3).

Емпіричний аналіз зі статті [16] призвів до двох припущень:

1. такі показники вказують на те, що автомобіль на дуже короткий проміжок часу потрапляє в короткочасну невагомість (наприклад під час входу в яму на дорозі);
2. можливо використовувати значення всіх трьох осей для встановлення припущення, про яму на дорозі, без прив’язки акселерометра до певного положення.

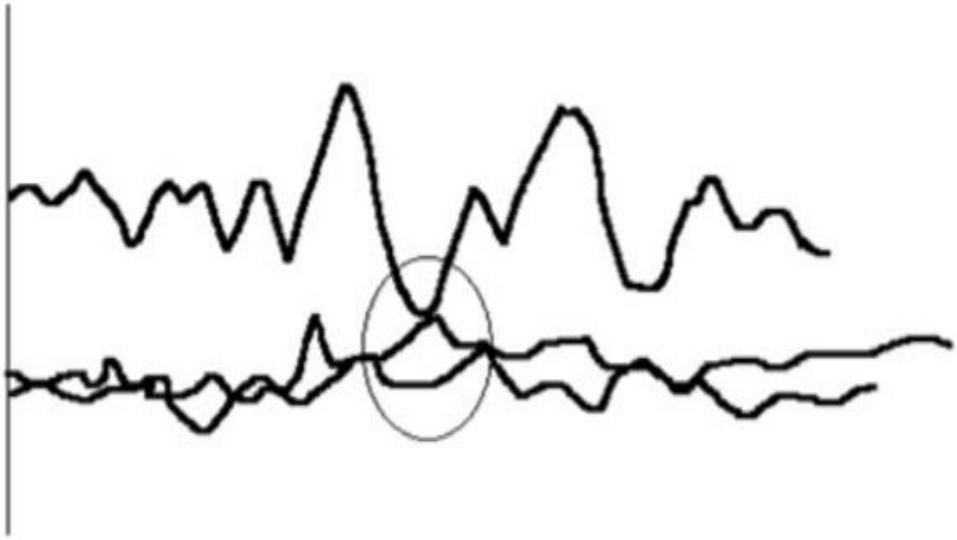


Рис. 1.3. Пошук ям за допомогою алгоритму G –ZERO [16]

Алгоритм Pothole Patrol

В статті [17] розглядається система визначення якості дорожнього покриття під назвою P² (Pothole Patrol). Дана система є більш складною, оскільки вона вміє класифікувати різні стани дороги, а саме:

- рівна дорога;
- дорожні компенсатори;
- дорожні «рельси»;
- ями;
- каналізаційні люки.

Класифікація відбувається на базі показників отриманих з осей X та Z акселерометра, та швидкості автомобіля, для збільшення точності перевірок (рис. 1.4). В ході дослідження було проведено тестування на різних типах доріг, та різних покриттях в місті Бостон, та на його околицях.

В тестуванні брали участь працівники таксі з різною манерою їзди на машинах різного року випуску. В кожній машині було встановлено комп'ютер Soekris 4801 з ОС Linux, і приєднаним до нього трьохосьовим акселерометром, розміщеним в однаковому положенні відносно осей. Під час тестування було перевірено 2492 км унікальної дороги за декілька тижнів.

Результат дав приголомшливий результат – лише 0.2% похибки при виявленні ям.

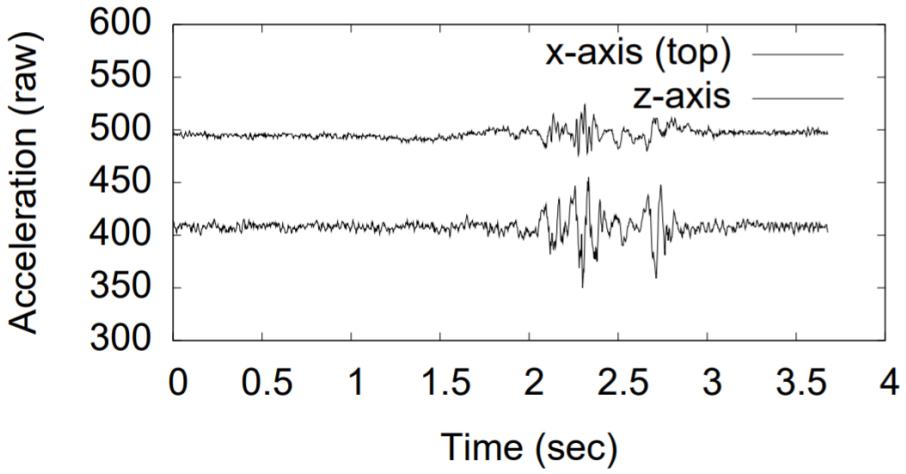


Рис. 1.4. Типове представлення ями для алгоритму P² [17]

Алгоритм на базі машинного навчання

Інший підхід до рішення представлений в статті [18]. В ній розглядається варіант знаходження ям для велосипеду, використовуючи вбудований акселерометр телефона. Головною відмінністю даного методу від інших є класифікація за допомогою машинного навчання. Класифікатором є наївний байєсовий класифікатор, навчений на комп'ютері за допомогою різних відрізків короткої довжини.

Під час руху система розділяє дорогу на короткі ділянки і перевіряє отримані данні з класифікатором онлайн і визначає цей відрізок або як рівну дорогу, або як дорогу з ямами. Результатом дослідження визначено точність на рівні 82,857%.

1.3 Постановка завдання

В рамках бакалаврської роботи, згідно обраної теми, необхідно розглянути існуючі апаратні технології в предметній області, для вибору оптимального апаратного забезпечення, провести порівняння технологічних рішень. Також, опираючись на обрані технології обрати необхідне програмне

забезпечення, для реалізації програмної частини, обрати відповідне середовище розробки.

Наступним кроком є розробити необхідний алгоритм для виконання поставленої задачі. В ході розробки алгоритму реалізувати методи для зчитування даних з датчиків, обробки даних, пересилання їх в хмарні сховища, обробки результату. Створити апаратну схему пристрою.

Завершальним етапом роботи є реалізація системи знаходження ям на дорогах за допомогою датчика лінійного прискорення з наступними функціями: зняття показників з датчика, фільтрування шумів та похибок, пересилка даних в хмарні сховища з подальшою обробкою. За результатами створеної системи необхідно провести тестування.

Також, однією з поставлених цілей є розробка методичних матеріалів для розгляду частин дипломної роботи, як набору лабораторних робіт.

Висновки до розділу 1

1. Було виявлено, що системи пошуку ям на дорогах є актуальною, тому, що вона може принести велику користь, як для дорожніх служб, так і для кожного автомобіліста по всьому світу. Вирішенням цієї задачі займаються ремонтні компанії, виробники автомобілів та просто ентузіасти по всьому світу.
2. Розглянуто кардинально різні підходи вирішення задачі пошуку ям на дорогах. Виявлено їх основні переваги та недоліки. Представлені варіанти існуючих алгоритмів фіксації показників.
3. Під час огляду літературних джерел визначено, що основним і оптимальним підходом для вирішення даної задачі є використання акселерометра, через його простоту, наглядність результату та низьку ймовірність відмови.
4. За результатами огляду літератури були поставлені задачі бакалаврської роботи, а саме розробка програмного та апаратного забезпечення для пошуку ям на дорогах, як частини IoT системи.

РОЗДІЛ 2
ПІДГОТОВКА ДО РОЗРОБКИ ПРОЕКТУ

Оскільки в ході огляду літератури розглянуто декілька різних способів вирішення поставленої задачі, то виникає необхідність вибору технологій, які будуть використовуватись саме в цьому проекті(апаратні засоби, методи збору та обробки інформації, мова, та середовище розробки). Згідно з цим твердженням виникає необхідність провести огляд існуючих технологій.

2.1 Огляд апаратного забезпечення, використаного під час реалізації проекту

Зараз на ринку існує безліч різноманітних пристроїв та датчиків, які використовуються в проектах різного ступеню складності. В цьому підрозділі необхідно розглянути та обрати необхідне апаратне забезпечення, для подальшого використання в проекті.

2.1.1 Вибір технології для отримання лінійного прискорення, для пошуку ям на дорогах

Основним способом отримання значення лінійного прискорення є використання акселерометрів. Загалом акселерометри поділяються на три види: одно-, двох-, та трьох осьові в залежності від того в скількох площинах виміряється значення лінійного прискорення. Найпоширенішими є трьохосьові, оскільки вони є більш універсальними і підходять для вирішення задач в якій з будь якою кількістю осей. [19]

Дані прилади поділяються на :

- п’єзоелектричні;
- поверхнево інтегральні;
- об’ємні інтегральні.

Розглянемо всі види датчиків, для визначення оптимального.

П’єзоелектричний датчик

П’єзоелектричний датчик представляє з себе стержень, який постійно створює тиск на п’єзокристал. Під час вібрації відбувається вироблення електричного струму, виміряючи покази якого відбувається визначення показників прискорення. Недоліком таких датчиків є високий рівень похибки і неможливість випуску в промислових масштабах тому, що кожен датчик відрізняється своїми параметрами конфігурації та дуже залежить від температури, вологості та тиску.

Об’ємні інтегральні датчики

Об’ємні інтегральні датчики складаються з двох пластин кремнію сплавлених між собою. На пластині розміщені три тонкі кремнієві смужки, до яких приєднана інерційна маса. Вагомий важок приєднаний до кремнієвої рамки з одного боку. Ці короткі балки обладнані декількома імплементованими п’єзорезисторами, що утворюють напівміст. Коли датчик починає рухатись у просторі з певним прискоренням, маса рухається в різні сторони, згинаючи балки і викликаючи деформацію п’єзорезисторів. Завдяки такій конструкції, датчик і розміщена за межами кристалу електронна схема обробки сигналів генерують під час руху, електронний сигнал напругою від 50-100 мВ. Інтегральні датчики лінійного прискорення з об'ємною конструкцією мають свої недоліки. По-перше, вони складні у виробництві, оскільки формування об'ємних елементів на платі складно з’єднуються з плоскими інтегральними платами. По-друге, датчик великого розміру. Зі зменшенням розмірів кристала підвищується його механічна міцність і знижується вартість. В з датчику об'ємною конструкцією тільки на розміщення чутливого елемента необхідна велика площа на кристалі. Якщо окрім самого датчику розмістити ще і схему формування сигналу, то зайнята площа збільшиться щонайменше в два рази. [20]

Поверхнево інтегральний датчик

Найкращою з точки зору ціна/якість є поверхнева інтегральна схема. Таку технологію в загальному називають MEMS (Мікроелектромеханічна система). Загальний принцип таких датчиків побудований на розрахунку зміщення інерційної маси відносно стану спокою та перетворення цього значення в пропорційний електричний сигнал. Серед всіх можливих варіантів ємнісний спосіб перетворення є найбільш точним, тому він є найбільш поширеним. Суть роботи такого датчику наступна – більшу частину кристалу такого датчика займає схема формування сигналу і лише трохи місця займає невеликий датчик прискорення розміщений в самому центрі кристалу, як представлено на рис.2.1. Датчик складається з великої кількості конденсаторів з повітряним діелектриком, обкладки яких зроблені з плоских смужок полікремнієвої плівки. Нерухомі обкладинки цього конденсатора - стержні, розміщені на 1 мкм вище від поверхні кристала в повітрі на полікремнієвих стовпчиках-анкерах, з'днаних з кристалом на молекулярному рівні. Рухомі стрижні між обкладинками змінюють конфігурацію конденсатора, що призводить до зміни показників напруги. Саме показники напруги на конденсаторі використовуються для розрахунку показників прискорення. [21]

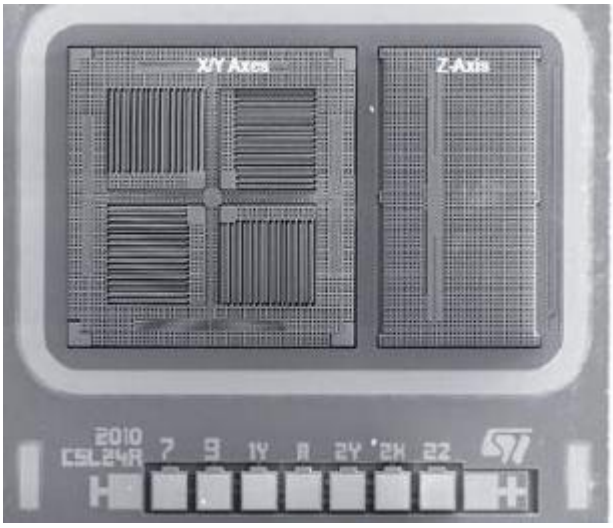


Рис. 2.1 Внутрішня структура трьохосьового МЕМС акселерометра

Основними плюсами цього акселерометра є простота в побудові і малі розміри. Ці два аспекти призводять до того, що ціна на такі датчики є низькою і вони доступні для вільного продажу. Також варто відзначити, що дана технологія показує гарні показники точності, достатні для нашого проекту. Загальне порівняння оглянутих технологій представлено на табл.2.1.

Таблиця 2.1

Порівняння типів датчиків лінійного прискорення

Тип	Точність	Ціна	Спеціалізація	Особливості
П’єзоелектричні	Низька	Низька	Удари, вібрації	Показники приблизні. Висока чутливість до температури та тиску.
Поверхнево інтегральні	Середня	Середня	Нахили, вібрації, інерційні сили	Низький рівень шуму. Великий розмір. Складно налаштовується.
Об’ємні інтегральні	Висока	Висока	Нахили, вібрації, інерційні сили	Малий розмір, завершена конструкція.

Виходячи з всього вищевказаного, оптимальним для поставлених задач є використання поверхнево інтегрального датчику.

2.1.2 Порівняння відомих акселерометрів

Визначившись з технологією для зняття показників лінійного прискорення необхідно порівняти існуючі поверхневі інтегральні датчики та обрати модель, яка буде задовольняти наші потреби.

Для огляду розглянемо датчики компанії STMicroelectronics. Ця компанія є провідною на ринку мікроелектроніки. Комплектуючі створені цією компанією використовуються у більшості проектів з використанням таких технологій у світі, тому буде досить легко знайти необхідну документацію для роботи з такими датчиками. Для всіх акселерометрів даної компанії характерним є схожа функціональна структура(рис.2.2). Значення з аналого-цифрового перетворювача потрапляють у вигляді 8 старших та молодших бітів. Після сумування даних значень після проходження відновлюючого фільтра стають доступними для отримання за допомогою інтерфейсів, таких як I2C або SPI.

Розглянемо три моделі для порівняння, а саме LSM303DLHC, LSM9DS1 та LIS3DSH.

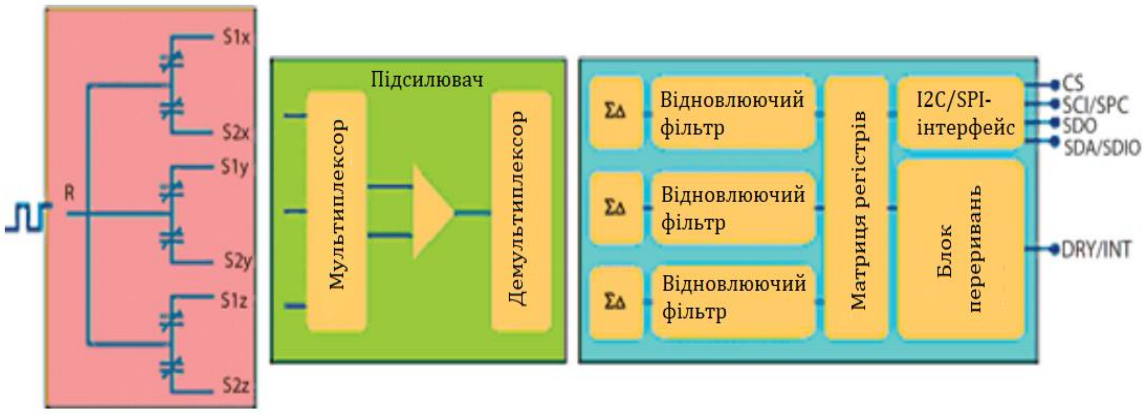


Рис. 2.2 Функціональна схема цифрового акселерометра [22]

LSM303DLHC

Згідно до офіційної документації датчика LSM303DLHC [22] це ультра компактний акселерометр та магнітометр. Основною його відмінністю від аналогів є суміщення разом двох датчиків на одному кристалі, окрім наявного акселерометра, він може орієнтуватись в просторі за допомогою магнітометра. Акселерометр на базі цього датчика підтримує розрахунок прискорення для трьох осей. В його можливості входить налаштування граничних розрахунків від ±2g до ±16g, де g рівне прискоренню вільного

падіння. Передача даних з датчика відбувається по шині за допомогою інтерфейсу I2C (Inter-Integrated Circuit). Шина складається з двох двонапрямлених ліній – послідовної лінії даних та послідовної лінії тактування, приєднаних до живлення, та керованих через відкритий колектор. Підтримується як стандартний (100кГц), так і швидкий режим роботи (400кГц) шини. Датчик працює при напрузі 2.16-3.6В. Розміри цього датчика всього 3*5*1мм.

LIS3DSH

Наступним до розгляду пропонується МЕМС акселерометр LIS3DSH. В порівнянні з попереднім датчиком, цей є повністю спеціалізованим саме на визначення лінійного прискорення. Він також трьохосьовий і підтримує виміри в діапазонах 2-16g, проте він вільно налаштовується на проміжні значення, для економії заряду. Основними його плюсами є низька енерговитратність і малі розміри. Датчик здатен працювати за напруги всього в 1.71 В. Загалом розмір датчика всього 3*3*1мм, що майже вдвічі менше за попередній. Даний акселерометр надстабільний – він витримує перевантаження до 10000g. [23]

Для передачі даних підтримується одразу два інтерфейси – вже раніше згаданий I2C, а також SPI - послідовний периферійний інтерфейс. Цей інтерфейс створений для двостороннього спілкування периферії та мікроконтролера. SPI - синхронний інтерфейс, тому передача даних відбувається на тактовій частоті мікроконтролера, який його використовує. В даному інтерфейсі завжди є один ведучий елемент, зазвичай мікросхема, і один, або декілька ведених елементів (периферія). Обмін даними відбувається по 4 лініям зв’язку, як представлено на рис.2.4.:

- MOSI (Master Out Slave In) — вихід мікроконтролера – вхід периферії. Він служить для передачі даних з ведучого до веденого;

- MISO (Master In Slave Out) — вхід мікроконтролера, вихід периферії. Служить для передачі даних в зворотному напрямку, від веденого до ведучого;
- SCLK (Serial Clock)— послідовний тактовий сигнал, який надсилає тактовий сигнал від мікроконтролера до всіх периферійних пристроїв;
- CS (Chip Select) — вибір веденого елемента, з яким відбувається обмін даними.

Принцип інтерфейсу полягає в обміні даними між зсувними регістрами мікроконтролера та периферії. Для вибору необхідного пристрою для обміну подається сигнал по лінії CS. Далі за допомогою зсуву регістрів останні 8 біт регістру введеного заносяться в перші 8 біт ведучого і навпаки через лінії MISO та MOSI, так як це представлено на рис.2.3. Частота обміну рівна тактовій частоті ведучого пристрою. [24]

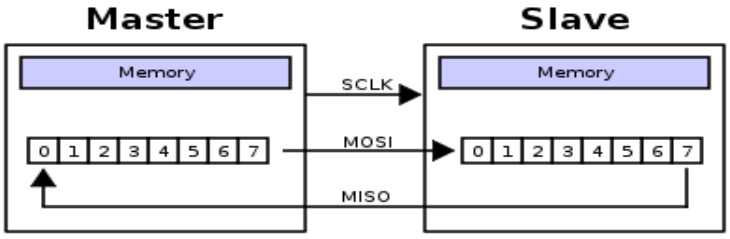


Рис. 2.3 Структурна схема обміну інформації в SPI [24]

Головною перевагою даного інтерфейсу є можливість обміну даними мікросхеми одразу з декількома периферійними пристроями, що є зручним для IoT систем, що складаються з великої кількості датчиків.

LSM9DS1

Останнім для розгляду обрано датчик LSM9DS1. Поміж всіх інших він має найбільший функціонал. Такий вид периферійних пристроїв називають інерціальними модулями. Таку назву він має завдяки наявності на одному чіпі розміром 3*3.5*1 одразу трьох датчиків: акселерометра, гіроскопа та

магнітометра. Проте задля такої різноманітності розробникам довелося пожертвувати певними характеристиками, в порівнянні з LIS3DSH. По-перше, підвищилась його енергозатратність. Для роботи датчик потребує живлення 1.9-3.6В. Також зменшилась можливість налаштування порогів вимірювання. Це призводить до зменшення точності вимірювання проміжних результатів. Єдиним незмінним залишається підтримка інтерфейсів SPI та I2C. [25]

Таблиця 2.2

Порівняння акселерометрів компанії STM32

Назва	Функціонал	Вольтаж	Способи передачі даних	Межі вимірювання	Чутливість
LSM303DLHC	Акселерометр, магнітометр	2.16-3.6В	I2C	±2g/±8g/±16g	0.09g
LIS3DSH	Акселерометр	1.71-3.6В	SPI та I2C	±2g/±4g/±6g /±8g/±16g	0.06g
LSM9DS1	Акселерометр, магнітометр, гіроскоп	1.9-3.6В	SPI та I2C	±2g/±4g /±8g/±16g	0.061g

Виходячи з усього вищесказаного, та опираючись на данні представлені в табл.2.2, оптимальним вибором для поставленої задачі є використання LIS3DSH, оскільки його функціоналу достатньо для нашої задачі і він є надзвичайно енергоекономним. Датчик LSM9DS1 без сумніву має достойні показники для своєї функціональності і надає можливість використання додаткових параметрів, але так як в даній роботі використання таких датчиків не є необхідними, то більш доцільним є використання спеціалізованого датчика.

2.1.3 Вибір технології зняття показників з периферійних пристроїв

Не останню роль в даній роботі займає вибір апаратних засобів для обробки інформації. Це безпосередньо вливає на швидкість та якість обробки даних та складність розробки системи. Розглянемо, які пропозиції існують на ринку в даний момент.

Arduino

Перше, що приходить в голову, коли йде мова про прості та недорогі плати для отримання даних з електронних датчиків, це використання однопроцесорних цифрових плат Arduino.

Вибір такої технології є оптимальним для початківців, з відсутністю навичок в комп'ютерній електроніці, спираючись на його простоту. Плати Arduino легко програмуються, завдяки посиланню чітких команд мікроконтролеру, розміщеному на платі. Для цього розробники плати створили спеціалізовану мову програмування Arduino та середовище розробки Arduino IDE. [11]

Завдяки свої простоті, плата Arduino зарекомендувала себе невід'ємною частиною багатьох проектів, починаючи від простих студентських проектів до складових частин IoT систем. Плата складається з мікроконтролера Atmel AVR з вбудованим завантажувачем та обв'язки для приєднання додаткових елементів. Тактування відбувається на частоті 8 або 16МГц з напругою 3.3-5В. Плата має велику кількість аналогових та цифрових входів та виходів. Характеристики плат різняться в залежності від комплектації.

Програми для Arduino пишуться на мові C++ з додатковими функціями для реалізації вводу/виводу з контактів плати. За допомогою середовища розробки, написані вами програми з легкістю зможуть бути вшитими в плату в один клік.

Для приєднання додаткових електричних деталей не знадобиться паяльник. Зібрати проект можна на зручній макетній дошці, без жодних труднощів. [12]

Основними перевагами Arduino є:

- низька ціна, в порівнянні з аналогами;
- крос-платформність (Середовище розробки працює на операційних системах Windows, Macintosh OSX, Linux);
- простота в програмуванні;
- програмне та апаратне забезпечення з відкритим кодом та можливістю розширення.

Raspberry Pi

Raspberry Pi – це комп'ютер, в якому всі складові розміщені на одній платі розміром 8,5*5,5 см. Спектр його використання залежить лише від того, які периферійні пристрої приєднати до нього. Виробники цифрових пристроїв використовують його для аудіосистем, метеостанцій та навіть окремих комп'ютерів. До нього можна приєднати всі необхідні для роботи частини (мишку, клавіатуру, монітор, тощо).

Raspberry Pi включає в себе процесор з частотою 700 МГц, відеокарта VideoCore IV і 512 мегабайтів оперативної пам'яті. Накопичувачем в даному комп'ютері являється SD карта. Офіційною мовою розробки є Python. На такий комп'ютер можна встановити навіть дистрибутив Linux. [13]

В якості процесору на Raspberry Pi виступає 4-ядерний ARM Cortex-A53. Він представляє з себе 32-бітний RISC процесор, який широко використовується в портативних пристроях. Основним плюсом даної архітектури є використання енергоефективних технологій, що і стає головним плюсом для розробників пристроїв, для яких головним є економія енергії.

Мікроконтролери STM

Мікроконтролери STM32 по праву вважаються найкращим рішенням для систем обробки сигналів в реальному часі. Це пов'язано з тим, що вони використовують цифрові сигнальні процесори ARM Cortex. Цифровий сигнальний процесор (Digital Signal Processor) - це мікропроцесор, спрямований на цифрову обробку сигналів, саме тому така архітектура поширена для вирішення задач, які виникають в цифровій обробці сигналів. Головними задачами цифрової обробки сигналів є:

- фільтрація;
- пошук сигналів;
- перетворення Фур'є.

Основним плюсом таких мікропроцесорів на відміну від інших є:

- виконання операції множення з накопиченням за один машинний цикл;
- апаратна реалізація циклічного виконання набору певних команд;
- одночасний доступ до декількох осередків пам'яті;
- векторно-конвеєрна обробка, використовуючи генератори адресних послідовностей. [14]

Також великим плюсом мікроконтролерів STM є низькі витрати енергії. В робочому стані на частоті те 84 МГц в режимі «Dynamic RUN» показники струму будуть не перевищувати 12 мА. [15]

В ході огляду літератури, нами було виявлено, що для вирішення завдання нам знадобиться використовувати мікроконтролер для отримання даних з акселерометра. На сьогоднішній день існує безліч варіантів різних мікроконтролерів. На сьогоднішній день існує безліч варіантів різних мікроконтролерів. В нашій роботі вибір падає на представників компанії STMicroelectronics. По-перше, обраний нами акселерометр належить даній компанії. По-друге, ними було реалізована широка лінійка різноманітних мікроконтролерів, тому ми зможемо обрати саме такий, який задовольнить

наші потреби. По-третє, програмування таких мікроконтролерів інтуїтивно зрозуміле і виконується на мові високого рівня. Крім цього переналаштування на плату іншої моделі досить легке, і потребуватиме лише незначних змін в конфігуруванні. Для розгляду обрані плати сімейства STM32F.

Базовий огляд серії STM32F

Мікроконтролери, побудовані на базі мікропроцесора ARM Cortex-M4, серії STM32 F4 є провідною серією лінійки STM32. Порівняно з попередніми, вони володіють більшою продуктивністю, порівняно зі старішими версіями. Ці мікроконтролери виготовляються використовуючи власний метод ST Microelectronics ART Accelerator для підвищення результатів продуктивності мікроконтролерів створених на ядрі Cortex-M. Це допомагає досягти високих показників, а саме 225 DMIPS / 606 CoreMark і можливість використання флеш-пам'яті на частоті 180 МГц. Також, в контролерах лінійки розміщений модуль операцій для роботи з числами з плаваючою комою, який дає можливість використовувати їх для задач широкого профіля. Також мікросхеми цієї серії славляться низьким енергоспоживанням. Це пов’язано з динамічним споживання живлення. Завдяки такому методу витрати по живленню складають до 140 мкА/МГц для STM32F401 і до 238 мкА/МГц для STM32F42x/43x на їх максимальних частотах.

Мікроконтролери на базі STM32 F4 об’єднують в собі можливість роботи в реальному часі і продуктивністю сигнальних процесорів при обробці сигналів і реалізують клас сигнальних мікроконтролерів. Загалом в серії представлено п’ять класів контролерів, які повністю сумісні в плані програмного коду і роботи з периферією. [26]

Спільні характеристики для всіх контролерів серії:

- 32-бітний процесор ARM Cortex-M4 CPU;
- Підтримка DSP-інструкцій;

- AHB-матриця шин;
- Напруга споживання 1,8…3,6В;
- Вбудовані RC-генератори з частотою 16МГц и 32кГц;
- Зовнішнє джерело переривань від 4 до 26МГц;
- Засоби налагодження SWD/JTAG;
- 12-бітний цифро-аналоговий перетворювач;;
- Сімнадцять 16- та 32-розрядних таймерів;
- Підтримка комунікаційних інтерфейсів: I2C, USART, SPI, I2S;
- Порти USB 2.0 FS/HS OTG;
- Контролер SDIO;
- Вбудований генератор випадкових чисел;
- Робоча температура від -40 до 105°C.

Як видно з представленого на рис.2.4 всі представники серії мають дуже високу схожість між собою в основних характеристиках. Відрізняються лінійки контролерів в основному, лише додатковим функціоналом.

Головні відмінності класів серії:

- STM32F401– 84 МГц CPU/105 DMIPS. Головна відмінність полягає в найменшому енергоспоживанні та малих розмірах, в порівнянні від інших лінійок серії;
- STM32F4x5 – 168 МГц CPU/210 DMIPS. Має 1 МБайт флеш пам’яті з широкими можливостями сполучення та шифрування;
- STM32F4x7 – 168 МГц CPU/210 DMIPS. Має 1 МБайт флеш пам’яті. В наявності інтерфейс Ethernet MAC;
- STM32F427/437 – 168 МГц CPU/210 DMIPS. Має 2 МБайт флеш пам’яті. Представляє з себе об’єднання можливостей STM32F405/415 та STM32F407/417;
- STM32F4x9 – 180 МГц CPU/225 DMIPS. Оперує 2 МБайтами двухбанкової флеш-пам’яті з інтерфейсом SDRAM, дисплеєм TFT LCD, технологією Chrom-ART, яка допомагає досягти

більшої продуктивності і меншого енергоспоживання ніж у мікроконтролерів серій STM32F4x7/F4x5.

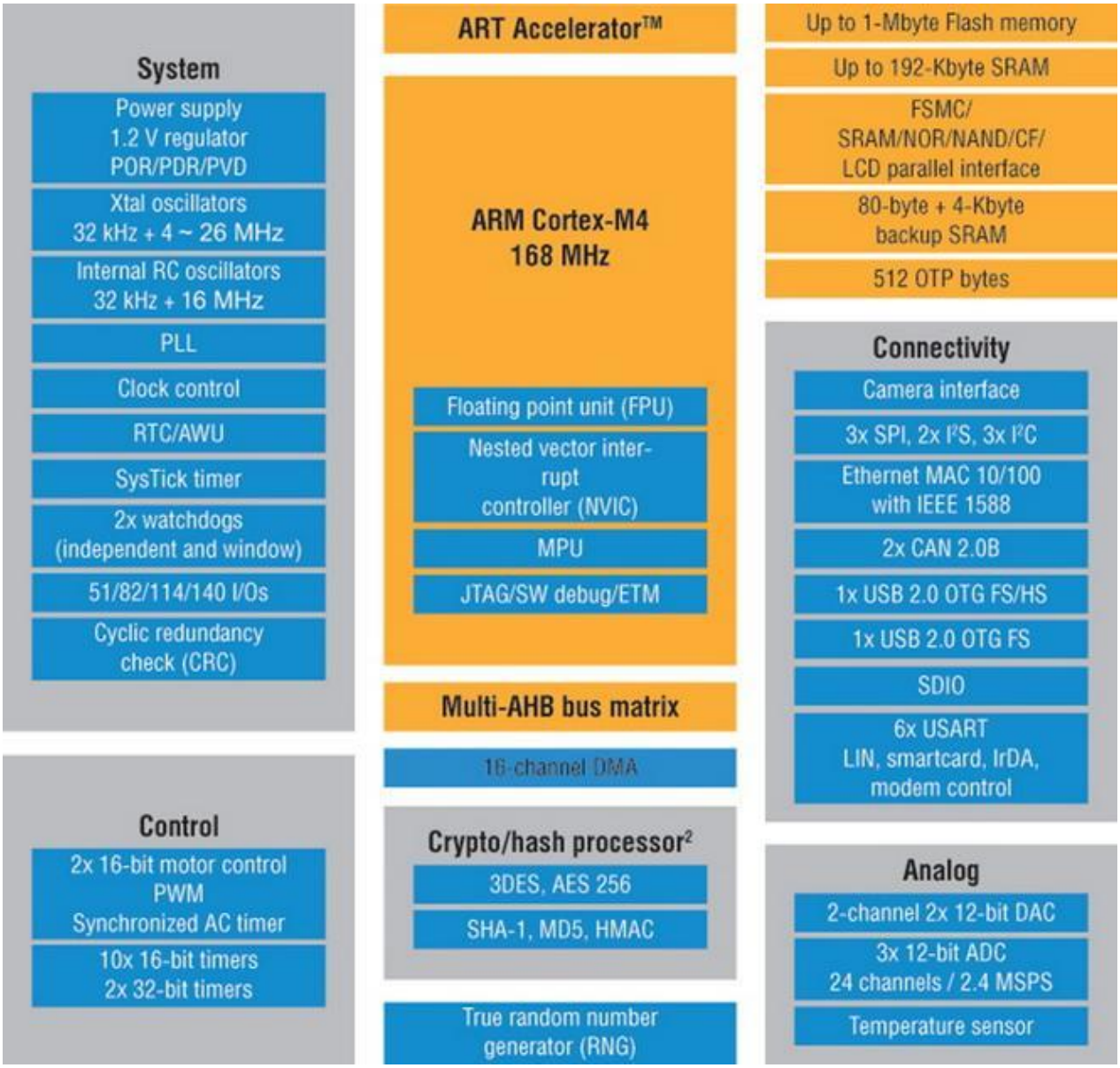


Рис. 2.4 Структура мікроконтролерів STM32F4 [26]

Варто також додати, що в даних мікросхемах добре реалізована робота з пам’яттю. На програмному рівні реалізовані API для C та C++, тому структура і топологія абсолютно неважливі для роботи програміста, оскільки відсутня необхідність мати справу напряму з самою архітектурою МК.

Особливістю системи STM32 є використання великої кількості портів вводу/виводу загального призначення. В загально прийнятому їх називають GPIO (General purpose input-output). Під назвою загального призначення мається на увазі, що стан виводів встановлюється програмно, і вони не є вхідними чи вихідними сигналами якогось вузла мікроконтролера. В системі STM32 порти 16-ти розрядні. Порти між собою абсолютно незалежні і їх стан встановлюється для кожного окремо. Кожен вихід може бути налаштований на такі режими:

- Input floating - вхід без підтягуючого резистора;
- Input pull-up - вхід з підтягуючим резистором, підключеним до живлення мікроконтролера;
- Input pull-down - вхід з підтягуючим резистором, підключеним до загального проводу (землі);
- Analog - аналоговий вхід;
- Output open-drain - вихід з відкритим стоком. Функціонально аналогічний виходу з відкритим колектором;
- Output push-pull - активний вихід. При низькому логічному рівні видає напругу рівну нулю, а при високому – подається напруга, близька до живлення мікроконтролера;
- Alternate function push-pull - альтернативна функція виведення в звичайному (активному) режимі;
- Alternate function open-drain - альтернативна функція виведення в режимі відкритий стік.

Для захисту конфігураційних параметрів існує спеціальна система захисту. Для її активізації необхідно виконати певну послідовність дій над регістрами конфігурації. Розблокування даних регістрів можливе тільки після скидання конфігурації порту.

В усіх представниках цього сімейства розміщено три аналого-цифрових перетворювачів і два цифро-аналогових перетворювачів. АЦП

мають високу швидкість перетворення (2,4 МСемпла в одиночному режимі і 7,2 МСемпла – в потрійному режимі). Загалом існує 24 аналогових каналів.

ЦАП володіє роздільною здатністю в 12 біт. Перетворення відбувається в 8/12-бітовому форматі з можливістю вирівнювання по правому або лівому краю. Оскільки ЦАП містить два канали, то існує можливість формування стереосигналу.

Також варто розглянути систему відлагодження програм. Для підключення до налагоджувача мікроконтролер має в наявності чьотирьохпровідний JTAG-інтерфейс, та двопровідний SWD. Оскільки виходи даних інтерфейсів мультиплексовані між собою, то розробник сам може обрати яким інтерфейсом йому зручніше користуватись при програмуванні мікроконтролера. [26]

Після детального огляду всієї серії мікроконтролерів оптимальним варіантом обрано контролер STM32F407VG. При такому виборі, за помірну ціну, ми отримуємо налагоджуючу плату з програматором-налагоджувачем ST-Link. Для початку програмування плати необхідно лише підключити плату до комп’ютера за допомогою спеціального кабелю і встановите середовище розробки на ваш комп’ютер. В платі окрім всього втановлена різноманітна периферія, така як акселерометр, мікрофон, USB та аудіокодек з роз’ємом для навушників або колонок. Так як і всі його аналоги в лінійці STM32F4x7 він базується на 32-бітному ядрі RISC ARM®Cortex®-M4, що працює на частоті до 168 МГц. Ядро Cortex-M4 має арифметичний елемент, який підтримує всі інструкції для роботи з плаваючою точкою. Він також реалізує повний набір інструкцій DSP та блок захисту пам'яті (MPU), що підвищує безпеку програми [27]. Детальне представлення апаратних можливостей даного контролера представлено на рис.2.5.

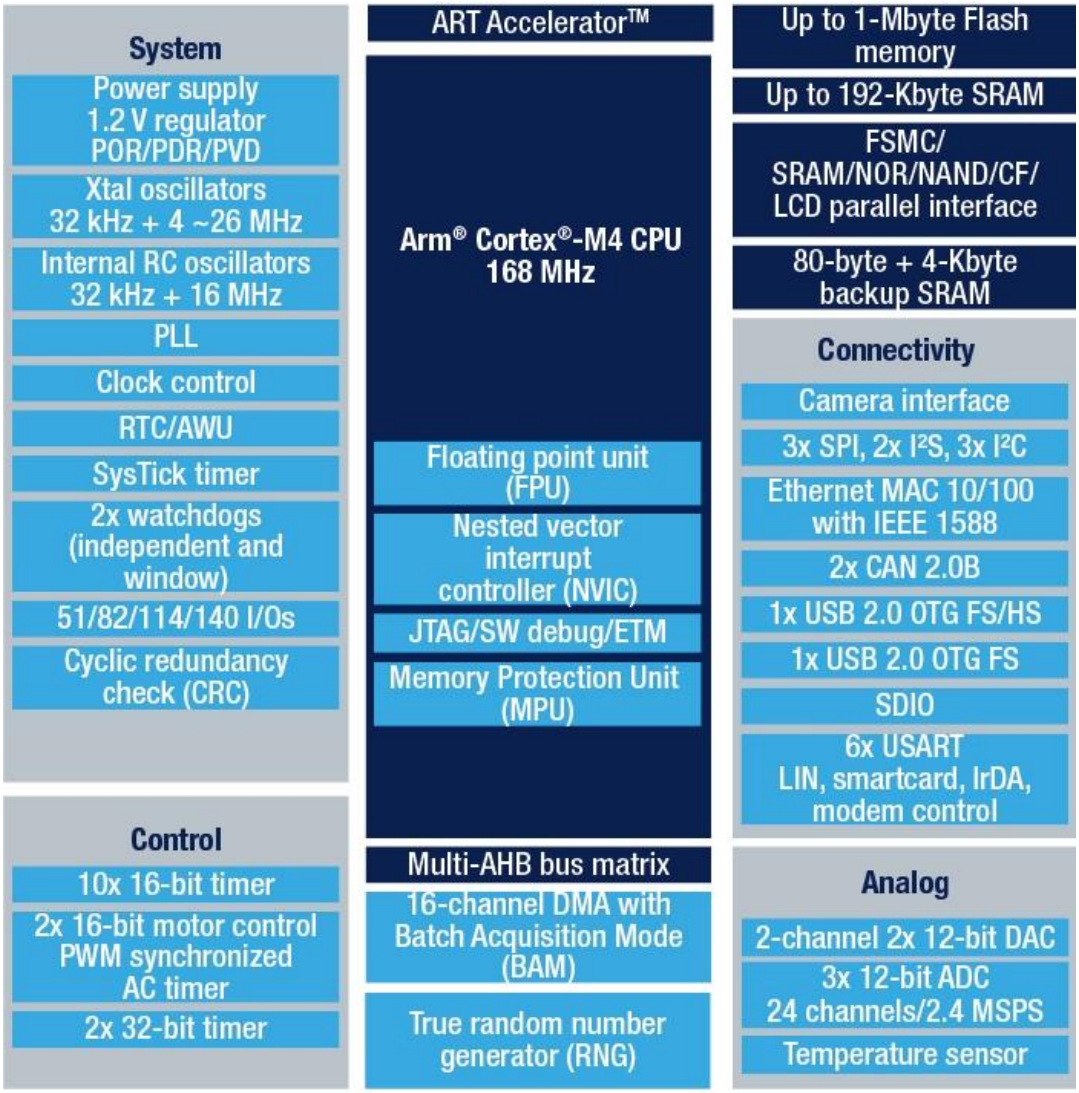


Рис. 2.5 Структурна схема мікроконтролера STM32F407 [27]

Одним із плюсів для його використання є підтримка інтерфейсу Ethernet, який в подальшому можна було б використовувати для обміну даними. Окрім всіх його плюсів, велику роль грає те, що на платі вже розміщений акселерометр LIS3DSH, який ми плануємо використати для нашої системи. Такий вибір допомагає уникнути роботи по сполученню датчика і мікроконтролера, оскільки за нас це вже зробив розробник. Також це економить кількість зайняти входів/виходів на платі, і надає можливість використати їх при подальшій розробці у разі потреби.

Отже, в цьому пункті мною було обрано апаратні засоби, які будуть використані в ході виконання роботи. Їх функціонал повністю відповідає потребам, які можуть виникнути в ході роботи.

2.2 Огляд програмного забезпечення, використаного під час реалізації проекту

Визначившись з необхідною апаратною базою проекту, необхідно розглянути і обрати мову опису апаратури, системи проектування, середовище розробки. Також слід розглянути методи взаємодії апаратних ресурсів та інтерфейси для виконання поставлених задач. Для будь-якого проекту надважливим до початку розглянути обрати такі важливі елементи як:

- Середовище розробки з компілятором;
- Система автоматизованого проектування;
- Мова розробки;
- Необхідні бібліотеки і ОС;

2.2.1 Використані системи програмування та налагодженя роботи мікроконтролерів

Для створення проектів будь якої складності з використанням мікроконтролерів необхідне середовище розробки Keil uVision. Дане програмне забезпечення представляє з себе набір утиліт для забезпечення всіх потреб під час написання програмного забезпечення мікроконтролерів.

Розробнику надається велика кількість функцій, представлених в інтуїтивно зрозумілому інтерфейсі (рис.2.6). Всі повсякденні функції представлені в контекстному меню, а менш вживані сховані для спрощення інтерфейсу, проте в будь якому проекті надана можливість використати весь потенціал модулів об’єднаних в одній програмі.

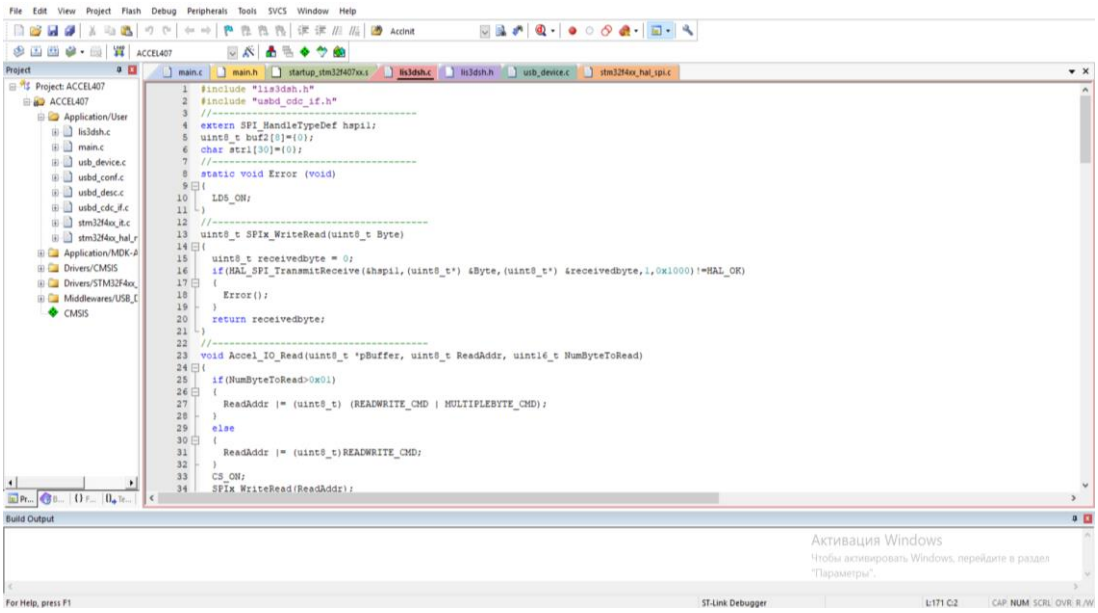


Рис. 2.6 Графічний інтерфейс середовища Keil uVision

Головними програмними засобами Keil uVision є:

1. База даних мікроконтролерів, яка містить в собі детальну інформацію про всі доступні до програмування пристрої. В цій базі зберігаються конфігураційні налаштування і посилання на додаткові джерела інформації з більш детальним технічним описом. Завдяки цій базі середовище автоматично встановлює всі опції мікроконтролера при доданні його до проекту.

2. Менеджер проектів, необхідний для з'єднання програмних модулів та файлів в групи для збереження ієрархічності та цілісності проектів. Це дає можливість краще орієнтувати в великій кількості файлів, та підтримувати порядок в робочі директорії.

3. Вбудований редактор, який спрощує роботу з текстовими файлами. Для більшої зручності є можливість використання багатовіконного інтерфейсу, налаштування зручного для роботи оформлення. В застосунку наявне відображення синтаксису програми за допомогою кольору та шрифтів. Також доступна функція редагування, одразу під час налагодження програми.

4. Засоби компіляції, асемблювання та компонування проекту для створення виконуваного модуля програми. В середовищі реалізована автоматична генерація асемблерних зв'язків, яка обробляє лише змінені файли, або файли, які залежать від таких після останньої збірки. Також в uVision реалізована функція оптимізації вихідного тексту проекту, що дозволяє досягти оптимального використання потужностей мікроконтролера шляхом багаторазової перекомпіляції коду. Компілятор працює з кодом на мові С, або асемблері для сімейств контролерів ARM, MSC51, C166 та багатьох інших.

5. Відлагоджувач-симулятор, який займається налаштуванням скомпільованої програми на віртуальній моделі мікропроцесора. В ході налагодження моделюється робота ядра контролера та периферійних засобів, таких як порти вводу/виводу, таймери, контролери переривань.

6. Додаткові функції, необхідні для спрощення повсякденних задач. Прикладами таких утиліт є:

- Source Browser, необхідний для пошуку програмних символів;
- Find in Files, пошук входжень тексту в файлах;
- Tools Menu, меню утиліт сторонніх програм;
- PC-Lint, аналізатор вихідного тексту, який відмічає можливі помилкові ділянки коду;
- Flash tool, який займається програмуванням FLASH-пам'яті мікроконтролерів. [28]

В даній програмі дуже зручно реалізований процес налагодження та компіляції програм. Середовище надає можливість виконання коду по рядкам, перегляду вмісту внутрішніх регістрів прямо під час налагодження, коли ваша плата приєднана до комп'ютера. Це значно спрощує процес пошуку помилок, тестування та налаштування мікроконтролера для коректної роботи.

STM32CubeMX

В ході роботи з будь яким мікроконтролером на початковому етапі, найголовнішим є приведення плати в робочий стан, її налаштування і конфігурування. Перед тим як приступити до будь якої розробки на платі потрібно встановити тактову частоту, активувати роботу необхідних інтерфейсів, визначити роботу для портів вводу/виводу та провести ще багато інших налаштувань. І така процедура може дуже сильно відрізнятися для мікроконтролерів різних виробників, а іноді навіть для моделей одного виробника. Для того, щоб робота розробника не затримувалася на виконанні рутинних команд налаштування, для пришвидшення та для впевненості в правильному налаштуванні саме обраного контролера компанія-розробник мікроконтролерів STM32 STMicroelectronics створила спеціалізовану програму для налаштування пристроїв власного виробництва під назвою STM32CubeMX.

STM32CubeMX це графічний інструмент для швидкого конфігурування мікропроцесорів та мікроконтролерів STM32, а також для генерації коду на мові С для ядра Arm Cortex-M або часткового дерева пристрою Linux для ядер Arm Cortex. Робота з даною програмою проста і інтуїтивно зрозуміла навіть для початківців. В STM32CubeMX створена база даних, в якій зберігаються базові налаштування для всіх існуючих мікроконтролерів STM32. Першим кроком в роботі з програмою є вибір мікропроцесора, який відповідає необхідному набору периферійних пристроїв. Вибір відбувається зі списку представлених варіантів, деталі і характеристики яких можна побачити в програмі, а у разі необхідності в більш глибокому розгляді реалізовані посилання на офіційну документацію. Після вибору апаратури з’являється можливість користувацького налаштування. Одразу на екрані перед вами з'являється схема всіх виходів плати, натискаючи на які можна одразу налаштовувати їх в різні режим. Після

конфігурування обрані входи змінюють колір, що дуже зручно для візувального сприйняття (рис.2.7).

Рис. 2.7 Робоче поле програми STM32CubeMX

Серед можливостей: налаштування портів вводу виводу, тактування всієї системи і окремих частин периферії, конфігурування вбудованого програмного забезпечення, такого як дерево налаштування тактування та калькулятор витрат енергії, а також утиліта для налаштування стеків середнього програмного забезпечення, на кшталт Ethernet, USB або TCP/IP. Всі ці та багато інших функцій легко реалізовуються завдяки графічному інтерфейсу, і як результат по завершенні налаштування і початку генерації ми отримуємо згенерований С код з обраними характеристиками та ініціалізованим ядром Arm Cortex, вже готовим до користування в середовищі розробки.

Ключові особливості програми:

- Інтуїтивний вибір мікроконтролера в графічному інтерфейсі;
- Простий у користуванні графічний інтерфейс користувача;
- Порти вводу/виводу з автоматичним вирішенням конфліктів;
- Функціональні режими програмного забезпечення та периферії з динамічною валідацією параметрів для ядра;

- Дерево тактування з динамічним преналаштуванням конфігурації (рис. 2.8);

- Потужність, згідно з прогнозованими результатами споживання;

- Генерація коду ініціалізації проекту, сумісного з компіляторами для плат STM32;

- Створення часткового дерева пристроїв Linux для ядра Arm Cortex-A;

- Наявність як автономного програмного забезпечення, що працює на операційних системах Windows, Linux та macOS або через плагін Eclipse. [29]

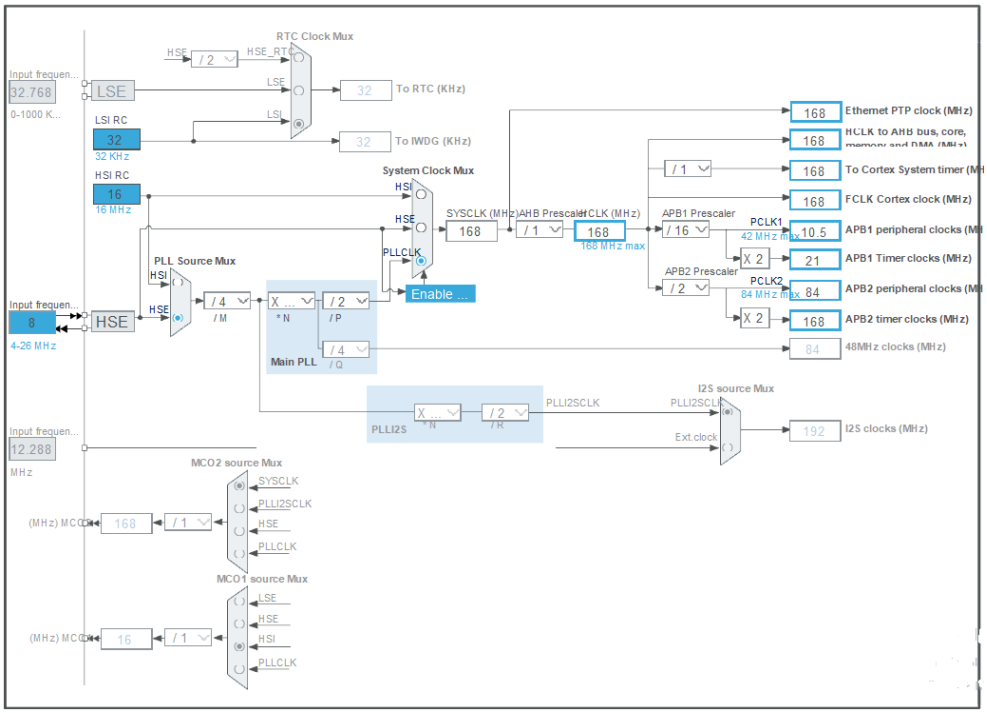


Рис. 2.8 Схема дерева тактування в STM32CubeMX

STM32CubeF4_FirmWare

Для роботи з мікросхемою з лінійки STM32F4x7xx реалізований набір бібліотек STM32Cube FirmWare F4. Саме в цій структурі розміщені всі можливі для використання бібліотеки, в залежності від апаратного забезпечення кожної мікросхеми з лінійки STM32F4.

Архітектура має три рівня реалізації, для опису свого рівня абстракції (рис.2.9):

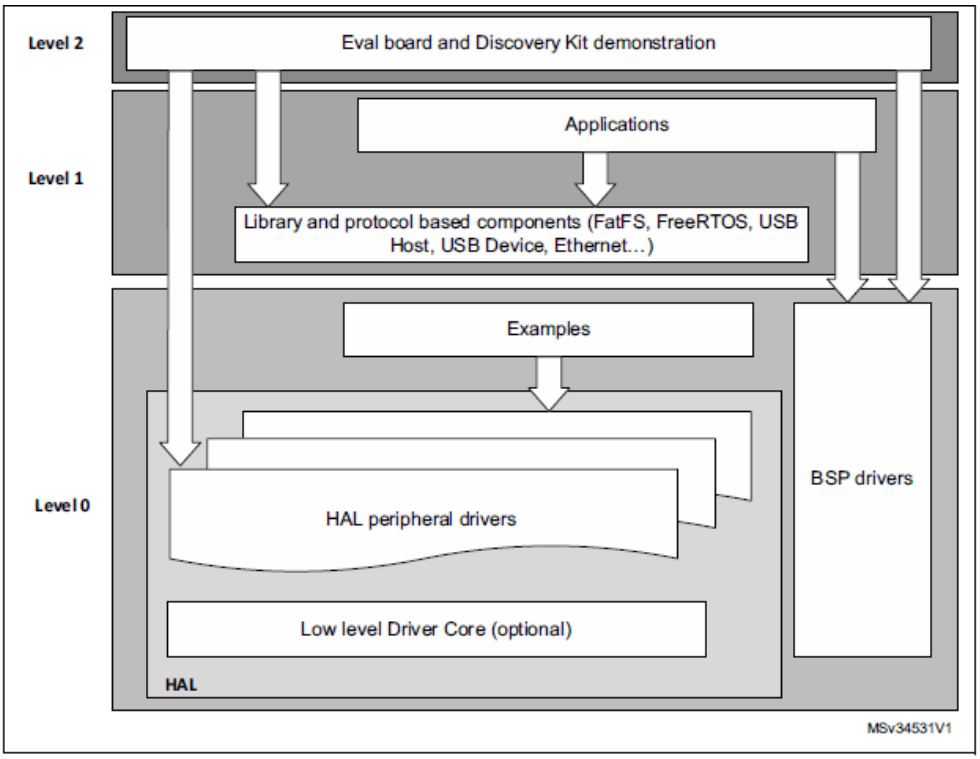


Рис. 2.9 Програмна архітектура STM32CubeF4_FirmWare [30]

Нульовий рівень абстракції

На нульовому рівні абстракції знаходиться Пакет Підтримки Платформи та рівень Апаратних абстракцій та низький рівень. Пакет Підтримки Платформи реалізує програмні інтерфейси для апаратних компонентів на платах, наприклад драйвери для аудіокодека, акселерометра, портів вводу/виводу, тощо. В даному пакеті також містяться драйвери для роботи з зовнішніми пристроями, не повязанными з STM32. Окрім цього в пакеті міститься драйвер BSP, для роботи з елементами розміщеними на конкретній платі. В ньому реалізовані зручні інтерфейси для роботи з периферією. З його допомогою, наприклад, запалення LED ліхтаря виконується одним рядком коду BSP_LED_On ().

Рівень Апаратних Абстракцій (HAL) складається з драйвера низького рівня і методів апаратних інтерфейсів, щоб взаємодіяти з верхніми шарами

(додатки, бібліотеки і стеки). Драйвери рівня HAL представляють з себе комплект універсальних, багатофункціональних інтерфейсів API, створених для взаємодії мікроконтролера з верхніми рівнями програмного забезпечення. Драйвери можуть мати як загальний, так і розширений API.

HAL створений для використання при програмуванні, коли необхідні функції виконуються верхнім шаром додатка, за рахунок застосування проміжного рівня HAL. При такій архітектурі програмування верхній рівень програми не прив'язаний до мікроконтролеру, тому що він звертається до мікропроцесора лише через бібліотеку драйверів. Використання такої структури в програмуванні дає можливість повторного використання коду в інших проектах і полегшує виконання переходу на інший пристрій STM32.

Драйвери HAL надають набір готових до використання API, які спрощують реалізацію роботи з мікроконтролером для користувача програми. Для прикладу, в бібліотеці HAL має великий набір методів для роботи з пристроями комунікації. В ній містяться інтерфейси API для ініціалізації і конфігурування пристрою, керування обміном даних за допомогою опитування, переривання або через DMA, і для управління помилками зв'язку.

API-інтерфейси драйверів HAL поділяються на дві категорії: загальні та розширені API. Загальні (generic) API реалізують спільні функції та інтерфейси для всіх пристроїв серії STM32. Розширені (extension) API складаються зі специфічних елементів та функцій, для спрощення роботи необхідних для окремих пристроїв. Драйвери бібліотеки HAL функціонально-орієнтовані, і не направлені на роботу внутрішніх периферійних пристрої. Наприклад, API таймера розділяється на категорії, в залежності від функціональності, як наявна у внутрішньому пристрої таймера: базовий таймер, захоплення, широтно-імпульсної модуляції (PWM).

Вихідний код бібліотеки драйверів розроблений відповідно до Strict ANSI-C. Це означає, що даний код є абсолютно незалежним і зрозумілим для

будь якого середовища розробки. Весь вихідний код перевірений інструментами статистичного аналізу CodeSonarTM. Код гарно чітко документований, інтуїтивно зрозумілий і є MISRA-C 2004 сумісним.

Також драйвери HAL підвищують рівень надійності вбудованого програмного забезпечення. В бібліотеках HAL реалізовано виявлення помилок під час виконання (run-time failure detection). Це означає, що HAL перевіряє отримані значення для всіх своїх функцій. Динамічне знаходження помилок під час виконання програми, також, сприяє прискоренню розробки призначених для користувача додатків і процесу налагодження.

Останнім для цього розділу є розгляд низького рівня (LL). Цей шар реалізує API низького рівня на рівні регістрів, для кращої оптимізації, жертвуючи портативністю. Використання даних функцій потребує глибокого знання мікропроцесора та його периферійних пристроїв на платі. Драйвери LL представляють з себе експертно-орієнтований шар, який знаходиться на найближчому до апаратного забезпечення рівні. На відміну від HAL, API LL не націлений на периферійні пристроїв де оптимізований доступ не є ключовим, або для периферійних пристроїв, які потребують складної конфігурації або стеків верхнього рівня

Перший рівень абстракції

На першому рівні абстракції розміщені компоненти проміжного рівня (Middleware components) - набір бібліотек, що охоплюють USB Host, STemWin, LibJPEG, FreeRTOS, FatFs, LwIP і PolarSSL. Взаємодія між компонентами цього шару здійснюються безпосередньо шляхом виклику API. Робота з драйверами низького рівня здійснюється за допомогою специфічних зворотних викликів.

Другий рівень абстракції

Цей рівень складається з одного шару, який реалізує графічне відображення в реальному часі першого рівня та шару низького рівня абстракції і додатків, які використовують периферійні пристрої для функцій підтримуваних налагоджувальним набором. [30]

2.3 Використання мови розробки С для програмування драйверів периферійних пристроїв

Розглянуті вище програмні та апаратні засоби передбачають використання мови С для розробки програмного забезпечення системи. Мова С зараз є однією з найпоширеніших мов програмування мікроконтролерів та периферійних пристроїв. На даному етапі, через появу нових мов програмування високого рівня мова С витісняється з багатьох сфер, проте вона все ще є незамінною для системного програмування. Мова С не має чіткої специфікації. Не зважаючи на невелику мовну базу і урізаний, на перший погляд функціонал, дана мова використовується як для низькорівневого так і для високорівневого програмування і є універсальною, на відміну від більш «складних» мов. Під час створення С розроблялася, як мова системного програмування. Її задачею мало бути використання при створенні UNIX компіляторів. Як вже вказано раніше, С не володіє великою функціональністю, проте така простота означає, що компілятори мови пишуться з легкістю, тому дана мова підтримується на безлічі платформ. Також мова володіє високим рівнем переносимості, хоча першочергово вона є низькорівневою, тому програми написані з використанням С гарно компілюються на пристроях з різними архітектурами.

Головною ціллю мови програмування С є спрощення написання великих програм зі зменшенням кількості помилок, в порівнянні з асемблером. Мова С підтримує принципи процедурного програмування, але

намагається по максимуму уникати речей, які можуть привести до додаткових розходів ресурсів, в порівнянні з мовами високого рівня.

Основні особливості мови С:

- Проста мовна база;
- Багато можливостей винесені в стандартну бібліотеку;
- Підтримка парадигми процедурного програмування;
- Система типів, які захищають від безглуздих операцій;
- Використання передобробки для пришвидшення роботи з однотипними операціями;
- Використання вказівників для прямого доступу до пам’яті;
- Невелика синтаксична база;
- Передача параметрів в функцію зі значенням без використання посилань;
- Передача параметрів через посилання з використанням вказівників;
- Наявність вказівників на функції та статичні зміни;
- Використання областей видимості імен;
- Наявність структур та об’єднань – створених користувачем збірних типів даних, якими можна маніпулювати як єдиним елементом. [31]

Але в той же час, на відміну від популярних мов програмування високого рівня в С відсутні такі можливості, як:

- Вкладені функції;
- Повернення більше одного значення з однієї функції;
- Співпрограми;
- Автоматичне керування пам’яттю;
- Засоби об’єктно-орієнтованого програмування;
- Засоби функціонального програмування.

Насправді, більшість відсутніх можливостей частково, або повністю імітуються вбудованими засобами або додаються за допомогою сторонніх бібліотек. Також, існують деякі компілятори, у яких реалізовані розширені

можливості мови, такі ,наприклад, як вкладені функції в компіляторі GCC. За допомогою тих чи інших можливостей можливими є використання співпрограм, багатопоточного програмування, збору сміття, мережевих функцій. Існує навіть методика, що дозволяє реалізовувати в мові С механізми ООП, що базується на фактичному поліморфізмі покажчиків в С та підтримці вказівників на функції.

Мова С була дуже добре прийнята розробниками, тому що допомагає швидко реалізовувати компілятори для різноманітних платформ, а також через свою наближеність до мов низького рівня дозволяє чітко уявляти, як саме буде виконуватись програма на комп'ютері. Також мова С дуже добре оптимізована. Завдяки своїй спорідненості з машинними мовами код написаний на С працює швидше і ефективніше, в порівнянні з мовами високого рівня. По рівню швидкодії С поступається лише оптимізованому коду, написаному на асемблері, оскільки він напряму надсилає команди до процесора. Оскільки зараз розвиток компіляторів та покращення процесорів відбувається в шаленому темпі, то написані асемблерні програми втратили свою цінність і майже не виграють в ефективності згенерованому коду компілятора мови С. Наразі С продовжує залишатися однією з найбільш ефективних мов високого рівня.

Структурно всі програми, написані мовою С схожі одна на одну. Вони представляють з себе набір файлів з кодом, які можуть бути перекомпільовані в об'єктні файли. Далі утворені об'єктні файли компонуються між собою та з файлами зовнішніх бібліотек, в результаті чого утворюється виконуваний файл, або бібліотека. Зв'язок файлів між собою та бібліотеками вимагає явного опису прототипів використовуваних функцій, зовнішніх змінних і необхідних типів даних в кожному файлі. Загально прийнятим рішенням є використання для таких завдань винесення таких даних в окремі заголовкові файли, які підключаються за допомогою директиви #include в разі необхідності певних даних, або функціональностей з інших файлів. Це надає

можливість організовувати, так звану, систему модулів. Модулем в такій системі найчастіше виступають:

- файли з вихідним кодом, для яких створений інтерфейс, у вигляді заголовкових файлів;
- об'єктна бібліотека з відповідними заголовковими файлами;
- інтерфейсна бібліотека – набір декількох заголовкових файлів;
- статична бібліотека або її модулі з відповідними заголовковими файлами;
- динамічна бібліотека або її окремі модулі з відповідними заголовковими файлами.

На етапі препроцесора директива #include лише підставляє текст іншого файлу в той в якому вона була викликана, отже в решті решт весь код з усіх модулів програми об’єднуються в одному файлі. Повторний виклик директиви непотрібний, оскільки він може викликати неочікувані помилки під час компіляції. [32]

Дана мова програмування широко використовується при розробці програм на прикладному рівні, в вбудованих системах, а також для написання високопродуктивних програм з високою критичністю в плані обробки помилок. Однією з причин, завдяки якій С досі популярний для низькорівневого програмування є можливість написання кросплатформних програм, які можуть працювати та різному обладнанні з різними операційними системами. Опираючись на все вищесказане вибір мови програмування зупинився саме на С, оскільки він оптимально підходить для роботи з платами STM32.

Висновки до розділу 2

В другому розділі було проведено детальний аналіз компонентів які необхідні для реалізації дипломного проекту, а саме:

1. Огляд та порівняння апаратного забезпечення (знайомство з мікроконтролерами STM32, та визначення необхідних периферійних пристроїв);
2. Огляд програмного забезпечення з огляду на обране апаратне забезпечення(розгляд мови програмування, середовища для програмування та засоби для налагодження та конфігурування програми);
3. Знайдено позитивні та негативні аспекти розглянутих технологій. Прийнято рішення використовувати мікроконтролер STM32F407vg, оскільки в ньому вже вмонтований цифровий акселерометр. Також мікроконтролери STM32 володіють високою потужністю при низькій енергозатратності. Розробку прийнято проводити в середовищі Keil uVision на мові C, через зручність та наглядність роботи, та можливість легкого завантаження написаних програм в мікроконтролер.
4. Зібрано всю необхідну інформацію для реалізації системи.

РОЗДІЛ 3
РОЗРОБКА ТА ОБГРУНТУВАННЯ СТРУКТУРИ ПРИСТРОЮ

3.1 Визначення структури пристрою

Згідно з інформацією, отриманою в ході розгляду технологій та методів отримання інформації з датчика лінійного прискорення було визначені всі складові, необхідні для реалізації системи.

Основна ціль для даного проекту є розробка програмного і апаратного забезпечення, яке можна буде інтегрувати в систему Інтернету речей з мінімальними переналаштуваннями. Причиною такого вибору є швидка популяризація розробки систем SmartCity. В таких системах працюють одразу десятки різноманітних датчиків, тому надзвичайно цінним є реалізувати систему таким чином, щоб розроблені драйвери могли бути використані, як частина більш широкого проекту. Саме тому в ході роботи були обрані саме такі методи та технічне забезпечення.

Загальна структурна схема пристрою представлена в Додатку 2. Перш за все, розроблена система буде реалізована на мікроконтролерах STM32, а саме STM32F407vg. Це зумовлено тим, мікроелектроніка даної компанії займає перше місце по поширеності по всьому світу. На платах STM розроблено більшу частину проектів пов’язаних з IoT та роботою з периферійними пристроями в цілому. Також надзвичайним плюсом є те, що незважаючи на високу відмінність багатьох мікроконтролерів коди написані для одного пристрою легко переконфігуровується для використання на інших пристроях компанії STM. Необхідно також відзначити і той факт, що в вище вказаній компанії були реалізовані зручні середовища розробки та конфігурування, що зменшує кількість втраченого часу при створенні проектів, на повторення однакових одного і того ж коду в різних проектах. Отже створення драйверів для периферійних пристроїв для використання на

базі мікроконтролерів є надзвичайно зручним і володіє можливістю перепрофілювання для різних плат.

Однак не зважаючи на це у відкритих джерелах досі мала кількість розробок, якими можна було б скористатися в рамках великих проектів, тому робота над такими проектами полягає в розробці драйверів, котрі вже були реалізовані велику кількість разів різними розробниками, а не в розробці нових алгоритмів, або методів для роботи з даними, які вони в подальшому отримають за допомогою своїх драйверів. Саме тому однією з цілей є створення драйверів як для розгляду та ознайомлення новачків з роботою таких пристроїв, так і для практичного користування в більш широких проектах.

Вибір саме плати STM32F407vg пояснюється тим, що в ній наявні всі необхідні елементи для вирішення поставленої задачі. Вона має низьку енергозатратність, при високих показниках продуктивності, володіє одразу декількома інтерфейсами обміну інформації, має в наявності велику кількість вільних портів вводу/виводу. Проте прив’язки до роботи саме з цією платою немає. Перенесення проекту можливе при переналаштуванні портів вводу/виводу, тактування та використанні вбудованих бібліотек для іншого мікроконтролера.

В роботі використано датчик лінійного прискорення LIS3DSH. Перш за все, вибір зупинився на ньому тому, що даний датчик вбудований в обрану плату, проте це не єдина його перевага. Він вузькоспеціалізований, а отже не витрачає зайву енергію при своїй роботі, що є дуже важливим для мікропроцесорних систем. Він має високу чутливість і гарно налаштовується на різні діапазони вимірів в залежності від поставлених задач. Також варто зазначити, що обраний датчик може комутуватись з системою за допомогою інтерфейсу SPI, що особливо зручно для систем IoT з великою кількістю периферійних пристроїв.

Для отримання даних з датчика було обрано використання шини SPI, як однієї з найзручніших, для отримання даних. Такий вибір пояснюється спрощенням роботи з даними, оскільки цей метод організовує швидкий синхронний зв’язок, а також може одночасно реалізовувати послідовний зв'язок між багатьма периферійними пристроями і одним мікроконтролером. Для виводу інформації було вирішено використовувати порт micro USB розмішений на платі STM32. Це дозволить в зручний і доступний спосіб отримати дані з плати на комп’ютер за допомогою послідовного COM-порту.

Всі процеси роботи на платі будуть реалізовані в середовищі Keil uVision на мові програмування C. Такий вибір пояснюється зручністю для роботи та високою внутрішньою оптимізацією згенерованого асемблерного коду, що означає високу швидкодію і мінімальні затримки в роботі драйверів. Конфігурування мікроконтролеру буде проведено в програмі STM32CubeMx, яка надає можливість швидкого налаштування всіх портів, бібліотек та інтерфейсів плати.

Кінцевим елементом є графічна програма, яка буде зчитувати інформацію з COM порту, обробляти отримані дані та виводити їх в зручному для людини вигляді. Дана програма буде написана на мові програмування Python за допомогою бібліотеки для роботи з послідовним портом pySerial та бібліотеки для графічного відображення даних mathplotlib.

3.2 Опис роботи пристрою в рамках системи Інтернету речей

Перш за все, при підключенні системи до живлення, відбувається перевірка працездатності системи, а саме відбувається опитування елементів системи на працездатність зчитуванням керуючих бітів, та перевірка, чи до заданих портів підключені периферійні пристрої, чи ні. У випадку помилки система припиняє свою роботу і запалює червоний ліхтар на платі, оскільки дані, які будуть отримані з системи з невірною конфігурацією будуть

хибними. У випадку, якщо перевірка пройшла успішно відбувається етап налаштування датчиків, ініціалізується шина даних SPI і відбувається перехід на другий етап налагодження.

В нашій системі, для правильності показників плата має бути розміщена горизонтально(паралельно дорожній смузі). У випадку, якщо пристрій розміщений в невірному положенні на платі вмикається один з 4 ліхтариків, які розміщені з чотирьох сторін навколо датчика. Світло ліхтаря вказує на те, що плата розміщена в неправильному положенні і потребує нахилу в ту сторону, з якої горить світло. Якщо всі чотири ліхтарі вимкнуті, це означає, що система готова до роботи.

По завершенню перевірок починається обмін даними між датчиком лінійного прискорення, та мікроконтролером. Зчитані з акселерометра дані приймаються датчиком і надсилаються на порт micro USB налаштований як послідовний COM порт.

При запуску програми Com Port Monitor на комп'ютері, на моніторі з'являється віконце графічного інтерфейсу з можливістю вибору Com порту і запуском відслідковування даних, які на нього приходять. Після вибору послідовного порту і натискання кнопки "Start" починається зчитування даних з порту і виведення значень у вигляді графіку. Комп'ютер який з'єднаний з мікроконтролером за допомогою кабелю USB зчитує дані з COM порту, обраного з доступних для роботи. Отримані дані приходять у шістнадцятковому вигляді і проходять процес форматування. Значення перетворюються в десяткову форму і приводяться до одиниці вимірювання $g=9.8м/с^2$. Після обробки, результати проходять через алгоритм пошуку ям, який визначає можливі проблемні ділянки дорожнього покриття. Паралельно з цим на моніторі комп'ютера відображається динамічний графік зміни вертикального прискорення. Графік представляє з себе набір двох статичних і однієї динамічної кривих. Дві лінії розміщені паралельно осі X і вказують гранично допустимі значення відхилення прискорення, які можна вважати

ямами. Третя крива демонструє оброблені показники прискорення по осі Z. У випадку знаходження різкого прискорення алгоритм розглядає таку ділянку, як потенційну яму і графік третьої кривої на цьому відрізку стає червоного кольору. У програмі доступні кнопки “Pause” та “Reset”, перша з яких зупиняє зчитування і ставить графік на паузу, а друга також зупиняє зчитування, але очищає поле на якому розміщений графік.

Таким чином результатом виконаної роботи буде графічне відображення показників прискорення транспорту з відображенням потенційних ділянок ям.

Висновки до розділу 3

В ході третього розділу було проведено підготовчу роботу до практичного виконання поставлених задач, а саме:

1. Обґрунтовано використання програмних та апаратних засобів використаних в системі;
2. Остаточно визначена структура проекту та її складові;
3. Обрано способи обробки і представлення отриманих даних;
4. Приведено опис роботи майбутньої системи;
5. Визначені методи та функції, які необхідно реалізувати для роботи системи.

З огляду на це, можна стверджувати, що всі підготовчі етапи завершено. Отриманої інформації достатньо і можна розпочинати програмну розробку системи.

РОЗДІЛ 4
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розробка апаратної частини проекту

Почати розробку проекту необхідно з програмного забезпечення апаратної частини проекту, а саме налагодження плати STM32F407vg, акселерометра LIS3DSH, та micro USB порту.

Конфігурування плати STM32F407vg

Першим кроком є налаштування плати для правильної роботи, а саме підключення необхідних бібліотек та інтерфейсів, ввімкнення портів, встановлення тактової частоти плати, тощо. Для цього скористаємось програмою STM32CubeMX.

Відкривши дану програму ми обираємо місце на диску, де буде зберігатися код нашого проекту та натискаємо “Next”. В наступному вікні нам потрібно обрати мікроконтролер, який ми будемо конфігурувати. В нашому випадку це STM32F407VGTx. Вікно вибору програми зображене на рис.4.1.

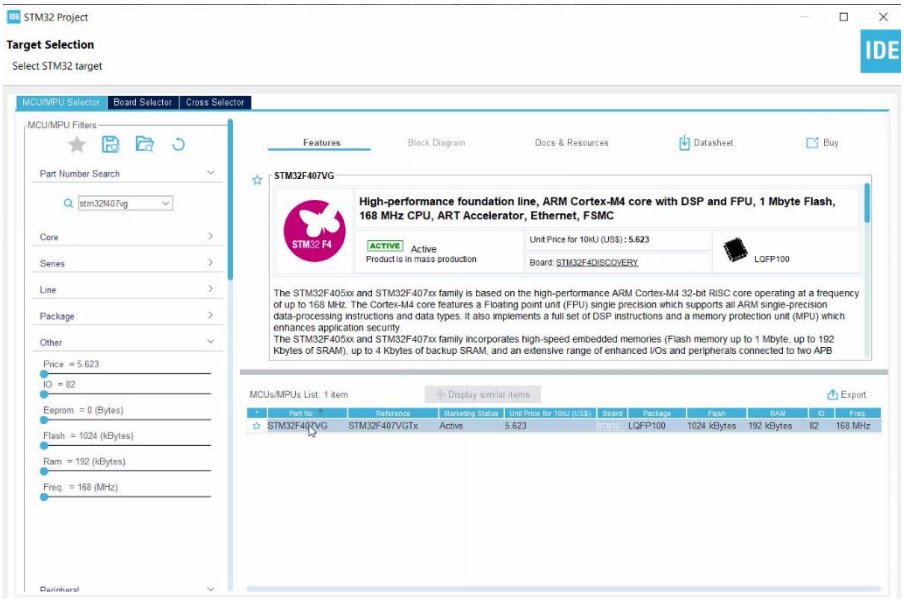


Рис. 4.1 Вибір мікроконтролера для налаштування

В наступному вікні встановлюється назва проекту, та мова на якій буде написано проект. Після цього знову натискаємо на кнопку “Next”, і перед нами відкривається головне вікно програми (рис.4.2).

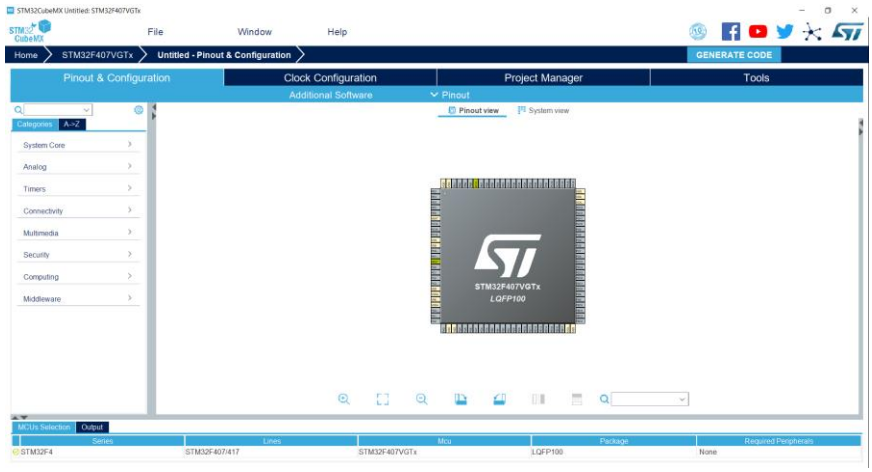


Рис. 4.2 Вигляд головного вікна програми STM32CubeMX

В даній програмі перед нами стоїть задача налаштувати плату на роботу з датчиком та micro USB портом. Для початку потрібно налаштувати тактування. Встановлюємо High Speed Clock в режим Crystal Ceramic Resonator в розділі SystemCore/RCC. Далі переходимо до вікна Clock Configuration та налаштуємо як представлено на рис. 4.3

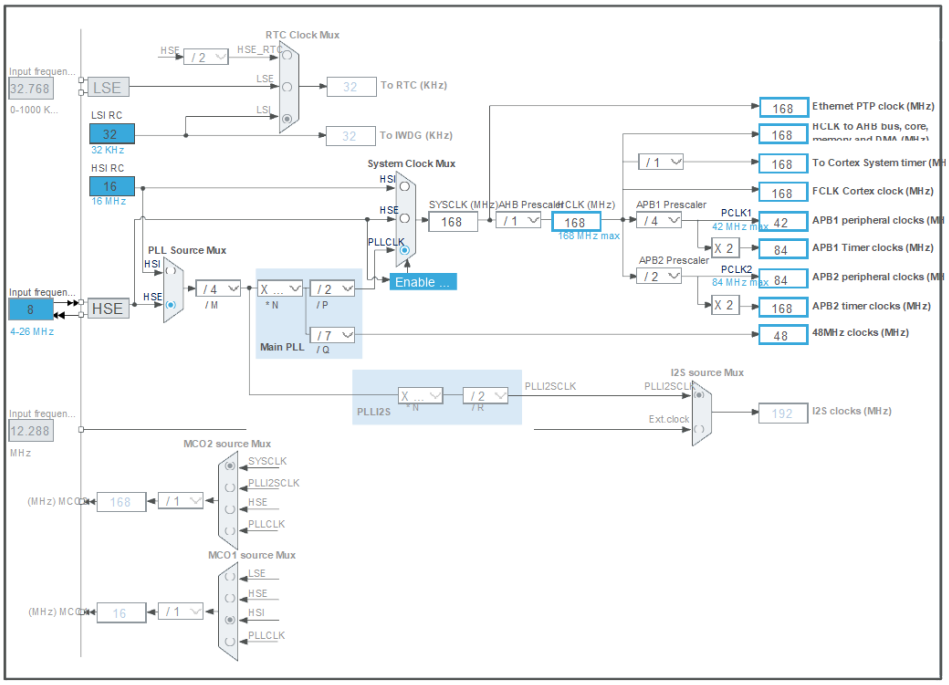


Рис. 4.3 Налаштування тактування мікроконтролера

Далі налаштуємо інтерфейси комунікації, які ми використовуємо в проекті, а саме SPI та USB. У вікні Pinout & Configuration в розділі Connectivity нас цікавлять пункти SPI1 та USB_OTG_FS. Пункт SPI1 в нашому проекті відповідає за обмін даними між акселерометром та мікроконтролером. В налаштуванні встановлюємо режим Full-Duplex Master, а в параметрах визначаємо дільник Prescaler рівним 16 для зменшення частоти передачі до допустимих показників передачі датчика. Далі переходимо до пункту USB_OTG_FS, налаштовуємо цей інтерфейс на режим Device_Only. Після цього в розділі Middleware з'явиться пункт USB_DEVICE. В цьому пункті нам потрібно визначити Class For FS IP як Communication Device Class (Virtual Port Com).

На цьому налаштування комунікації можна вважати завершеним, і можна переходити до налаштування портів вводу/виводу. Звернемось до офіційної документації мікроконтролера [33]. Згідно продемонстрованій схемі на рис.4.4 нам необхідно увімкнути порти PA5-PA7, PE3, PE1 та PE0 для роботи з акселерометром. Також увімкнемо порти PD12-PD15, які відповідають за LED ліхтарі, розміщені на платі. В результаті отримуємо таке налаштування портів, як на рис. 4.5.

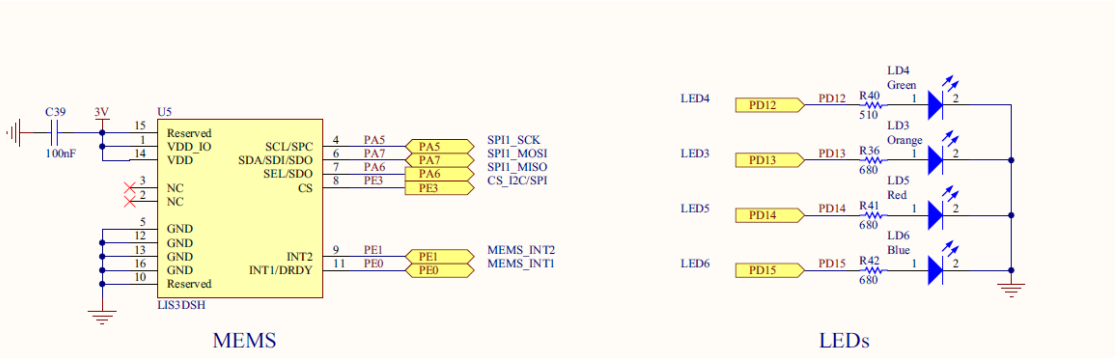


Рис. 4.4 Підключення портів вводу/виводу для датчика LIS3DSH та LED ліхтарів [33]

Далі заходимо в розділ Project Manager і в полі Code Generator/ IDE встановлюємо значення MDK-ARM. На цьому наше конфігурування завершується і можна переходити до написання коду. Натискаємо на кнопку Generate Code. Програма автоматично створює проект в середовищі Keil uVision і генерує код, який відповідає нашим налаштуванням.

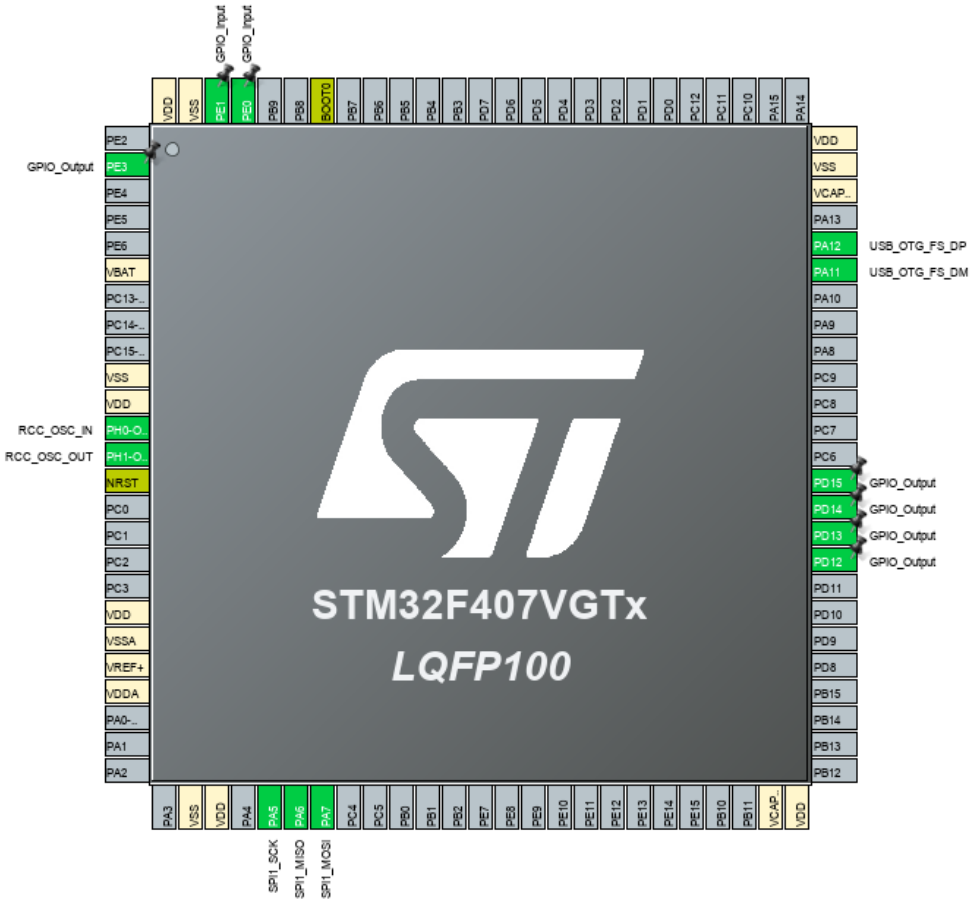


Рис. 4.5 Налаштовані порти вводу/виводу

Опис методів для роботи з акселерометром

По завершенні генерації ми отримуємо одразу декілька частково згенерованих файлів:

- main.c – головний виконавчий файл, який є тілом майбутньої програми. В ньому розташовується функція main(void), котра виконується при запуску програми, функції ініціалізації GPIO та

SPI, конфігурації системного тактування, та методи для контролю помилок;

- stm32f4xx_it.c – файл в якому зберігаються налаштування переривань;
- usbd_cdc_if.c – інтерфейс Com порту для USB з методами пересилки даних;
- пакет Drivers – пакет, в якому розміщені драйвери для стандартних бібліотек HAL та CMSIS.

Оскільки в проекті вже створений інтерфейс для обміну даними через послідовний порт, то ми можемо одразу створити прототип методу, який використовується для відправки даних в файлі usbd_cdc_if.c. Цей метод повністю задовольняє наші потреби, тому можна залишити його без змін.

uint8_t CDC_Transmit_FS(uint8_t* Buf, uint16_t Len);

Для роботи з акселерометром створимо файл lis3dsh.c та заголовковий файл lis3dsh.h. Код файлів представлений в додатку 4. В заголовковому файлі містяться всі макропідстановки необхідні для роботи з датчиком та прототипи функцій, для використання в інших файлах. Файл lis3dsh.c містить в собі методи для роботи з шиною SPI, ініціалізації датчика прискорення та обміну інформацією між акселерометром та мікроконтролером, а саме:

- SPIx_WriteRead(uint8_t Byte) – функція читання та запису 1 байта по шині SPI;
- Accel_IO_Read(uint8_t *pBuffer, uint8_t ReadAddr, uint16_t NumByteToRead) – функція читання NumByteToRead байтів даних з ReadAddr (адреси даних в датчику) в pBuffer завдяки шині SPI;
- Accel_IO_Write(uint8_t *pBuffer, uint8_t WriteAddr, uint16_t NumByteToWrite) – обернена функція запису NumByteToWrite байтів даних на адресу WriteAddr з буфера pBuffer;

- Accel_ReadID(void) – функція перевірки, правильності доступу до акселерометра;
- AccInit(uint16_t InitStruct) – функція налаштування контрольних регістрів датчика;
- Accel_GetXYZ(int16_t* pData) – функція зчитування даних з датчика;
- Accel_ReadAcc(void) – функція відокремлення показників прискорення по осях та перевірки правильності положення пристрою. Результатом роботи функції є відправка значення прискорення по осі Z по послідовному порту і запалення ліхтарів у випадку нерівного положення пристрою;
- Accel_Ini(void) – функція ініціалізації датчика. Збирає та надсилає конфігураційні дані в контрольні регістри.

Тепер для роботи з датчиком залишилося лише включити файли в файлі main.c директивою include, провести ініціалізацію датчика в коді даного файлу, та додати метод Accel_ReadAcc() в безкінечний цикл виконавчого методу main. На даному етапі ми маємо програмний код, який готовий до завантаження на плату. Для завантаження необхідно приєднати плату до комп'ютера AWG кабелем, натиснути на кнопку Build для компілювання проекту, та по завершенні, натиснути на кнопку Start Debugging Session. Після натискання програма завантажиться на мікроконтролер.

Вже зараз ми можемо побачити (рис. 4.6) перші результати роботи датчика, а саме налаштування правильного розміщення плати на поверхні. Для прикладу розмістимо ручку під платою в різних положеннях і подивимось на результат. При положенні А плата розміщена абсолютно рівно, тому її показники абсолютно вірні, а отже жоден з ліхтарів не запалюється. При положеннях Б та В плата розміщена під нахилом відносно осі X, а на положеннях Г та Д відносно осі Y. Це означає що при збільшенні

відхилення датчика від рівного положення по осях абсцис та ординат зменшується точність показників вертикальної осі. Саме тому, для забезпечення правильності значень, при невірному розміщення датчика вмикається ліхтарик, який вказує в яку сторону відбувається відхилення і затухає лише при вирівнюванні.

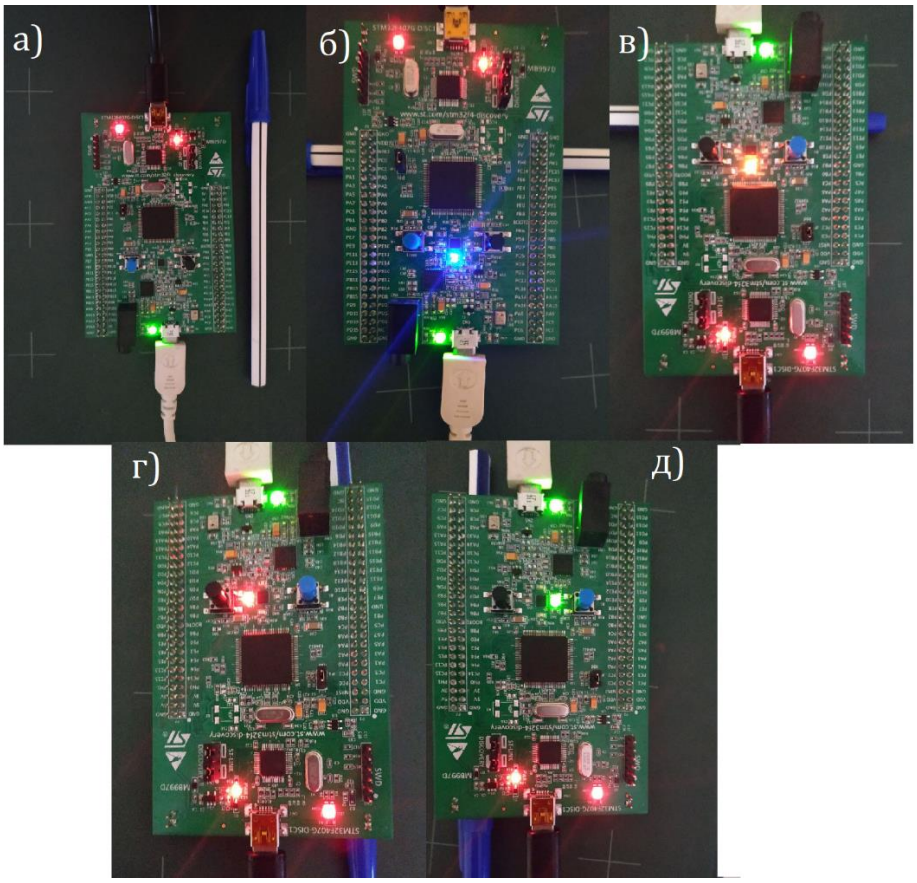


Рис. 4.6 Загорання ліхтарів при нерівному положенні плати

4.2 Розробка програмної частини проекту для обміну та обробки даних

Вже зараз на мікросхемі запалюються ліхтарі при відхилені плати в різні сторони і надсилає показники з акселерометра з Com порту через USB, але зараз ці данні немає чим зчитати, тому необхідно створити програмне забезпечення для зчитування даних і відображення їх в зручному для розуміння вигляді. Для рішення цієї задачі створимо програму написану мовою Python. Саме ця мова обрана тому, що для неї реалізовані зручні

бібліотеки для роботи з COM портом – pySerial, та для динамічного відображення графіків – matplotlib. В основі роботи цієї програми обрано такий алгоритм: програма буде представлена виконуваним файлом Com Port Monitor.exe. При запуску даного застосунку буде відкриватися вікно програми з меню вибору одного з доступних портів на комп'ютері, кнопками “Start” для початку читання даних та побудови графіку прискорення, “Pause” для зупинки зчитування та перебудови графіку, та “Clear” для очищення графіку та закриття порту. При натисканні на кнопку “Start” буде відкрито обраний порт для читання і розпочнеться етап обробки прийнятих даних. Перш за все дані будуть приведені до десяткового значення з плаваючою комою і відправлені в буфер. Дані зібрані в буфері відображаються на графіку в вікні програми.

Оскільки ми хочемо не лише побачити отримані значення, але і спрогнозувати подію потрапляння автомобіля в яму, то необхідно розробити алгоритм визначення ям опираючись на отримані дані. Алгоритм буде проводити дворівневу перевірку. Перш за все, при потраплянні машини в вибоїну машина значно змінює своє прискорення по вертикальній осі. З огляду на це, першим етапом перевірки є порівняння значення прискорення з пороговим допустимим значенням. Оскільки потрапляючи в яму прискорення набуває вигляд синусоїди - різко прискорюючись, а потім так само різко сповільнюючись, то необхідно встановити як верхній так і нижні поріг прискорення. По-друге, прискорення при потраплянні в яму зростає дуже різко, тому є можливість виявити тріщини, які можуть не призвести до високих показників, але зміна між послідовними замірами буде істотна для припущення, про наявність дефекту на дорозі. Отже, другим способом перевірки показників буде порівняння сусідніх значень в буфері показників. При знаходженні таких ознак відмітимо такі показники червоним кольором.

Визначившись з необхідним функціоналом створимо програму. Всі функції, створені для цієї програми містяться в одному файлі

ComPortMonitor.py (лістинг в додатку). В даному файлі знаходиться клас PortMonitor. В даному класі наявні такі методи:

- __init__(self)- конструктор для ініціалізації внутрішніх полів класу;
- zAxis(self) –метод для зчитування даних з послідовного порту;
- draw(self) – метод для обробки отриманих даних, та визначення дефектів дорожнього покриття;
- onButtonStartClicked(self,event) – метод для обробки події натискання на кнопку “Start”;
- onButtonPauseClicked(self,event) – метод для обробки події натискання на кнопку “Pause”;
- onButtonStopClicked(self,event) – метод для обробки події натискання на кнопку “Clear”;
- onRadioButtonsClicked(self, event) – метод для обробки події натискання на елементи списку для вибору порту;
- handle_close(self,evt) - метод для обробки події закриття програми;
- serial_ports(self) – метод для пошуку ввімкнутих портів у комп’ютері;
- mainLoop(self) – метод для відображення вікна програми з усіма елементами розміщеними на ньому. Викликається при запуску програми.

Для завершення програми і перетворення її в виконувану необхідно скористатися консольною командою *pyinstaller -F -w –i “icon.ico” main.py*. Давайте розберемося за що відповідають прапорці в цій команді:

- –F - вказує на очищення папки dist від сміття після створення exe файлу;
- -w - блокує появу консольного вікна при відкритті програми;
- -i - встановлює зображенням програми файл icon.ico;

- main.py - файл, який необхідно перетворити в виконуваний.

В результаті отримуємо виконуваний файл програми для визначення і графічного відображення ям на дорогах(рис. 4.7).

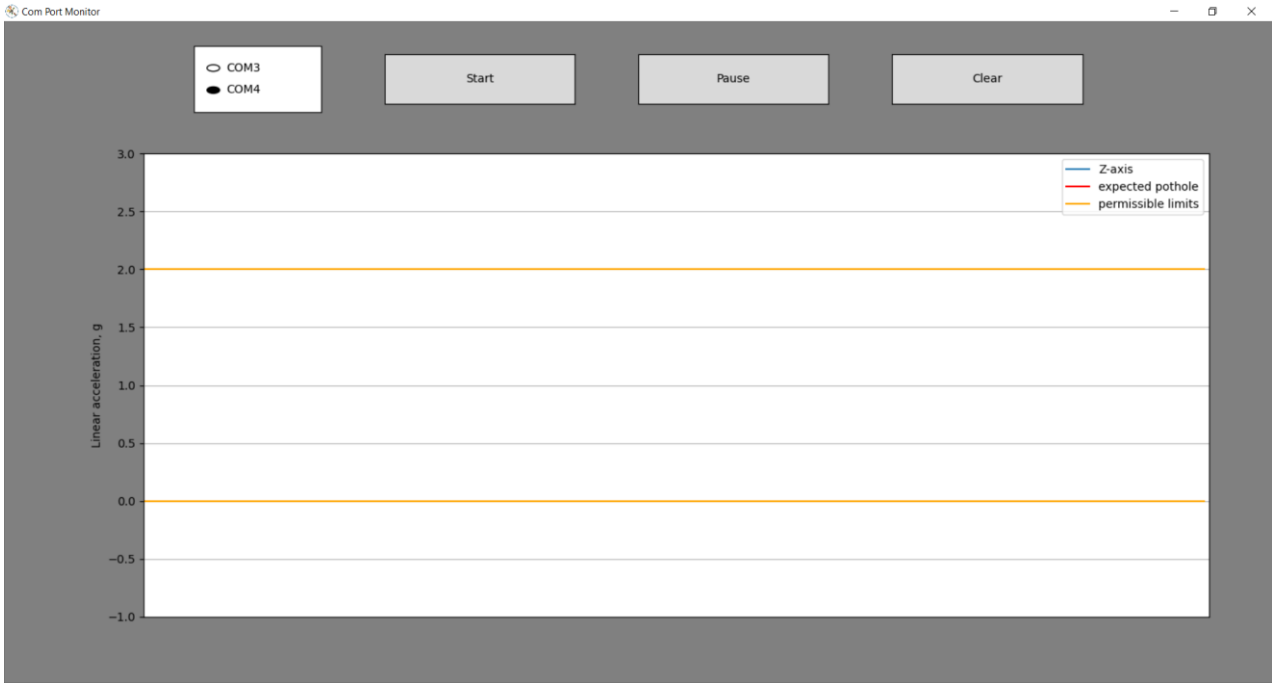


Рис. 4.7 Вигляд програми Com Port Monitor після запуску

4.3 Розгляд роботи інтерфейсу програми

В подальшому, для роботи в системі IoT, необхідність відображення показників за допомогою комп'ютерної програми зникне, тому інтерфейс програми повинен виконувати лише поставлені задачі і не потребує гарного оформлення.

Після запуску програми ми отримуємо вікно програми, яке представлено на рис. 4.7. На ньому розміщено три активні кнопки, поле вибору Com порту, з яким буде проводитись обмін даними та поле для відображення графіка. На графіку розміщена легенда та відмітки вертикальної шкали. Після натискання на кнопку "Start" починається відображення на графіку даних, отриманих з акселерометра(рис.4.8). Оскільки датчик в початковий момент часу знаходиться в стані спокою, то графік прискорення майже нагадує пряму лінію, і показники приблизно

близькі до прискорення вільного падіння. Давайте поглянемо на вигляд графіку під час руху (рис.4.9). Як ми можемо побачити, критичні ділянки відображаються на графіку червоним кольором, при чому, як ті, які перевищили допустимий поріг, так і ті, які дуже швидко прискорились.

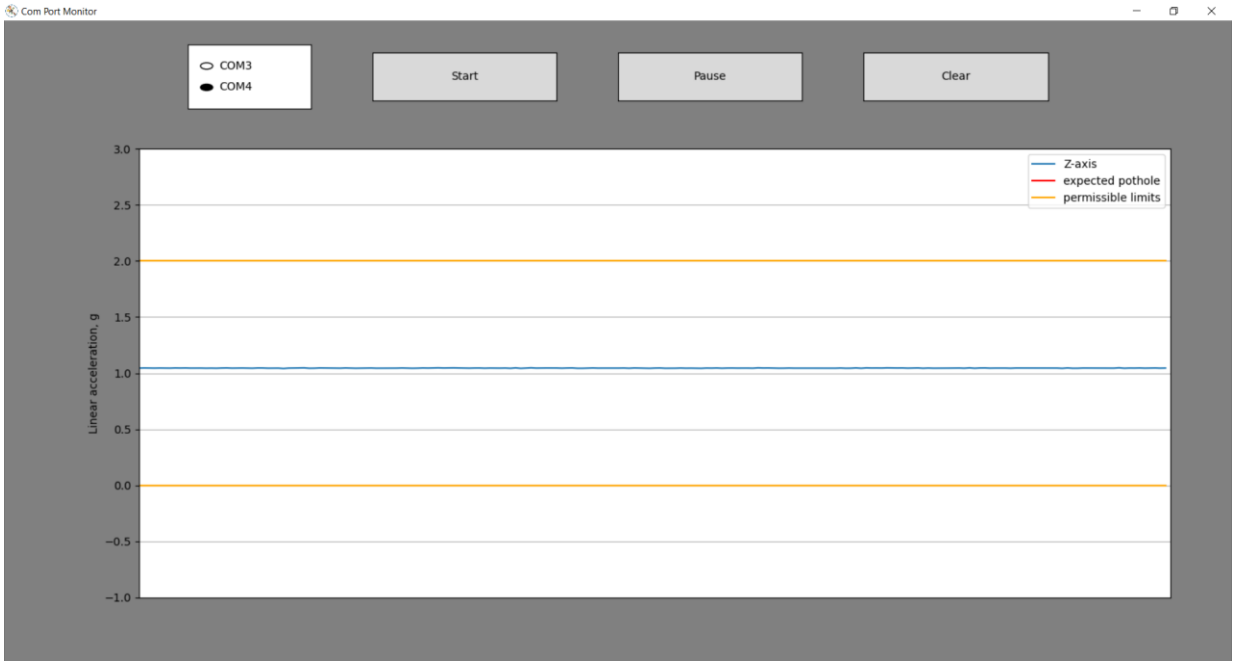


Рис. 4.8 Вигляд графіку прискорення при положенні в стані спокою

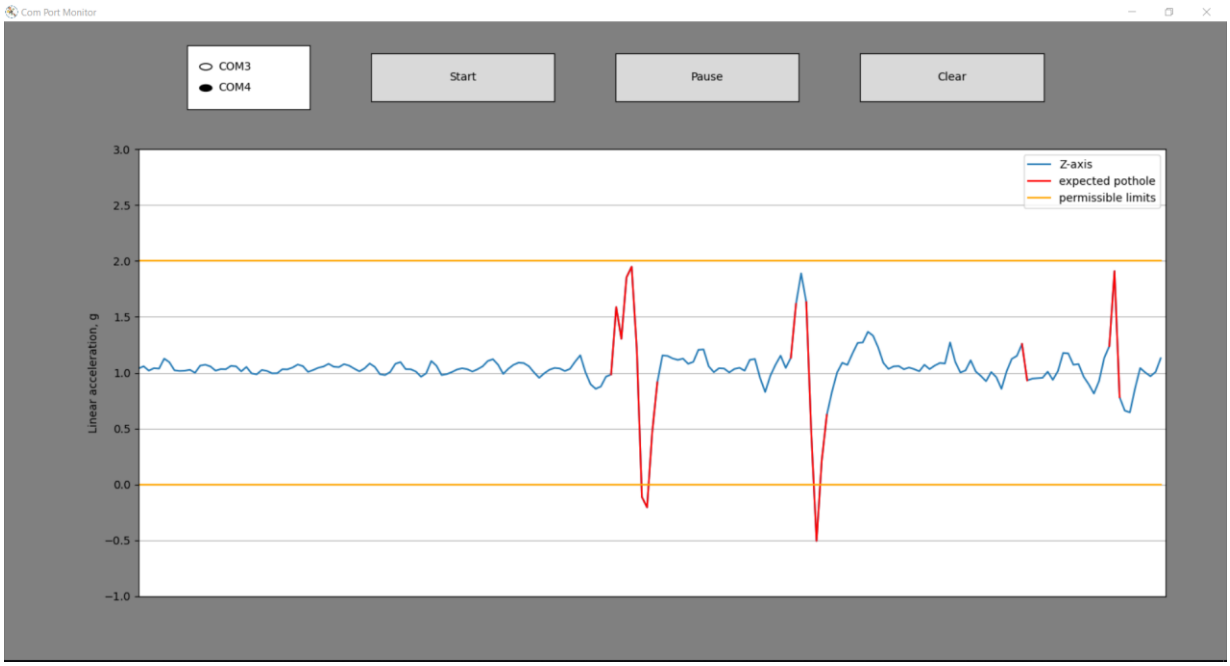


Рис. 4.9 Вигляд графіку прискорення під час руху

Зображення показане на рис 4.9 стало можливим завдяки натисканню на кнопку “Pause”. При натисканні на неї, зчитування зупиняється і графік завмирає в тому положенні, в якому він знаходився в момент натискання. У разі необхідності почати зчитування повністю заново на чистій робочій поверхні можна натиснути на кнопку “Clear”. Натискання на цю кнопку призведе до приведення вигляду робочого вікна в початковий стан, як під час запуску програми(див. рис. 4.7.)

Висновки до розділу 4

В ході четвертого розділу було проведено такі етапи роботи, як:

1. Розробка програмного забезпечення для апаратної частини IoT системи для виявлення ям на дорогах за допомогою акселерометра;
2. Розробка програми з графічним інтерфейсом для візуалізації отриманих результатів;
3. Опис методів та функцій, створених для реалізації системи;
4. Опис роботи створеної графічної програми Com Port Monitor;
5. Тестування отриманих результатів;

Опираючись на отриманий результат можна чітко сказати, що система має певні переваги, в порівнянні з аналогами, які існують в світі. Перш за все, створена модель має низький рівень споживання енергії, конкурентний для подібних мікроконтролерних систем і набагато переважаючий в порівнянні з система, як використовують мобільні пристрої. З недоліків можна визначити привязаність до правильного положення датчика, проте ця проблема вирішується жорсткою фіксацією, або доданням методу для перерахунку прискорення відносно статичної системи координат.

ВИСНОВКИ

Результатом бакалаврської роботи є система для визначення ям на дорогах з використанням акселерометра, як частина IoT система. Ця система була розроблена опираючись на огляд подібних пристроїв, їх переваг та недоліків.

Суть роботи пристрою – обробка даних, зчитаних з датчика лінійного прискорення для визначення ям на дорогах і представлення цієї інформації у зручному вигляді.

Основна ціль проекту – розробка аналізатора якості дорожнього покриття, як частини IoT проекту, який може бути максимально оперативно приєднаний до реальної багатофункціональної системи Інтернету речей. Паралельно з цим, стояла задача розробити документацію по роботі з платою STM32F407vg та MEMS датчиками на базі виконаного проекту.

На даному етапі, розроблено базову частину систему, яка правильно реагує на зміну положення та рух пристрою. Таке апаратне забезпечення в майбутньому може стати частиною більш крупного проекту, і збільшити свій функціонал. В подальшому можливе покращення процесу виявлення ям шляхом ускладнення алгоритму або використання методів машинного навчання.

В ході виконання дипломного проекту, мною були отримані та закріплені на практиці базові знання про архітектуру мікроконтролерів STM та методи обробки сигналів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Причины образования различных деформаций дорожного полотна и методы контроля за ними. [Електронний ресурс]: https://ceiis.mos.ru/presscenter/news/detail/7577271.html
2. Передвижная лаборатория НПО «Регион» | НПО «Регион» [Електронний ресурс]: https://nporegion.ru/laboratorii/
3. Алгоритмы детектирования разметки и дефектов дорожного покрытия [Електронний ресурс]: https://istina.msu.ru/publications/article/3717828/
4. M. Kass, A. Witkin and D. Terzopoulos “Snakes: Active Contour Models” International Journal of Computer Vision, vol. 1, pp. 321-331, 1987
5. Специальный ИИ помогает быстро ремонтировать дороги - 1Informer | новости, гаджеты, технологии [Електронний ресурс]: https://1informer.com/technologii/spetsialnyj-ii-pomogaet-bystro-remontirovat-dorogi-14569
6. UaRoads [Електронний ресурс]: https://uaroads.com/ua/index
7. Украинские программисты запустили сервис, собирающий информацию о ямах на дорогах | AIN.UA [Електронний ресурс]: https://ain.ua/2014/05/15/ukrainskie-programmisty-zapustili-servis-sobirayushhij-informaciyu-o-yamax-na-dorogax-dannye-peredadut-v-ukravtodor/
8. Ford предупредит водителя о ямах на дороге [Електронний ресурс]: https://chudo.tech/2017/02/22/ford-predupredit-voditelya-o-yamah-na-doroge/
9. Volvo расскажет другой Volvo о гололеде [Електронний ресурс]: https://chudo.tech/2016/11/25/volvo-rasskazhet-drugoj-volvo-o-gololede/
10. Jaguar и Land Rover научаться обнаруживать и избегать ямы на дорогах [Електронний ресурс]: https://adt.by/jaguar-i-land-rover-nauchatsya-obnaruzhivat-i-izbegat-yamy-na-dorogax/
11. Arduino - Introduction [Електронний ресурс]: https://www.arduino.cc/en/Guide/Introduction

12. Что такое Arduino: первые шаги в освоении электроники / Амперка [Електронний ресурс]: https://amperka.ru/page/what-is-arduino
13. RPi Hub - eLinux.org [Електронний ресурс]: elinux.org/RPi_Hub
14. FAQ: ЦСП – цифровой сигнальный процессор [Електронний ресурс]: http://fpga.in.ua/dsp/dsp-theory/faq-csp-cifrovoj-signalnyj-processor.html
15. Для интенсивных вычислений: STM32F401 с ультранизким динамическим потреблением [Електронний ресурс]: https://www.compel.ru/lib/56926
16. B. Lanjewar, Jyoti Khedkar, Rahul Sagar, Rasika Pawar, Kunal Gosavi. Survey of Road Bump and Intensity Detection algorithms using Smartphone Sensors. IJCSIT, Vol. 6, 2015. – URL: http://www.ijcsit.com/docs/Volume6/vol6issue06/ijcsit2015060659.pdf
17. Jakob Eriksson, Lewis Girod, Bret Hull, Ryan Newton, Samuel Madden, Hari Balakrishnan. The Pothole Patrol: Using a Mobile Sensor Network for Road Surface Monitoring. MobiSys, 2008. – URL: https://www.cs.uic.edu/~jakob/papers/p2-mobisys08.pdf
18. Marius Hoffmann, Michael Mock, Michael May. Road-quality classification and bump detection with bycicle-mounted smartphones. – URL: http://ceur-ws.org/Vol-1088/paper7.pdf (дата обр. 11.05.2016)
19. Акселерометр. Виды и типы. Работа и применение. Особенности [Електронний ресурс]: https://electrosam.ru/glavnaja/slabotochnye-seti/oborudovanie/akselerometr/
20. Интегральные акселерометры [Електронний ресурс]: http://www.compitech.ru/html.cgi/arhiv/02_01/stat_66.htm
21. МЭМС-датчики движения от STMicroelectronics: акселерометры и гироскопы [Електронний ресурс]: https://russianelectronics.ru/mems-datchiki-dvizheniya-ot-stmicroelectronics-akselerometry-i-giroskopy/
22. LSM303DLHC - STMicroelectronics [Електронний ресурс]: https://www.st.com/resource/en/datasheet/DM00027543.pdf

23. LIS3DSH - STMicroelectronics [Електронний ресурс]: https://www.st.com/resource/en/datasheet/lis3dsh.pdf
24. Интерфейс SPI [Електронний ресурс]: http://s-engineer.ru/interfejs-spi/
25. LSM9DS1 - STMicroelectronics [Електронний ресурс]: https://www.st.com/resource/en/datasheet/lsm9ds1.pdf
26. Обзор микроконтроллеров семейства STM32F4 [Електронний ресурс]: http://fpga.in.ua/dsp/dsp-theory/obzor-mikrokontrollerov-semejstva-stm32f4.html
27. STM32F407VG - STMicroelectronics [Електронний ресурс]: https://www.st.com/en/microcontrollers-microprocessors/stm32f407vg.html
28. Keil uVision - среда разработки программного обеспечения для микроконтроллеров [Електронний ресурс]: https://cxem.net/software/keil.php
29. STM32CubeMX – STMicroelectronics [Електронний ресурс]: https://www.st.com/en/development-tools/stm32cubemx.html
30. Description of STM32F4 HAL and LL drivers [Електронний ресурс]: https://www.st.com/resource/en/user_manual/dm00105879-description-of-stm32f4-hal-and-ll-drivers-stmicroelectronics.pdf
31. Guidelines for Choosing a Computer Language: Support for the Visionary Organization [Електронний ресурс]: http://archive.adaic.com/docs/reports/lawlis/k.htm
32. Керниган Б., Ритчи Д. Язык программирования Си — 2-е изд. — М.: Вильямс, 2007. — С. 304. — ISBN 0-13-110362-8.
33. UM1472 User Manual [Електронний ресурс]: https://www.element14.com/community/docs/DOC-50074/l/stmicroelectronics-user-manual-of-stm32f4discovery-stm32f4-high-performance-discovery-board

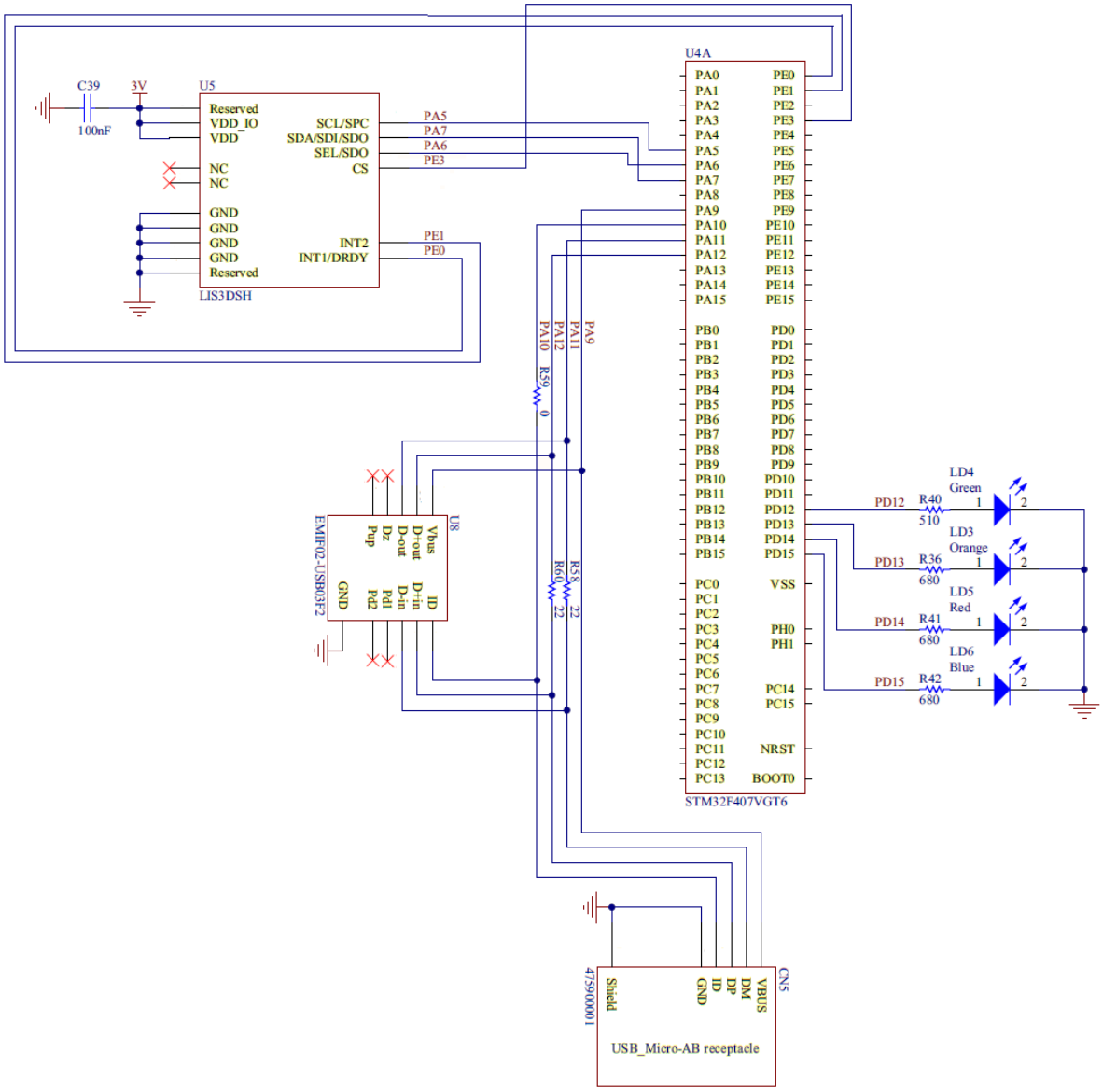
ДОДАТОК 1

IoT система з використанням акселерометра для SmartCity

Принципова схема апаратного забезпечення системи
ІАЛЦ.467100.004 Д1

Аркушів 1

Київ 2020 р.



					ІАЛЦ.467100.004 Д1
Зм.	Арк.	№ документа	Підп.	Дата	
Розробив		Копійка А.О.			ІоТ система з використанням акселерометра для SmartCity Принципова схема апаратного забезпечення системи
Перевірив		Ткаченко В. В.			
Н.контр.		Сімоненко В. П.			
Затверд.		Стіренко С. Г.			

Літ.	Аркуш	Аркушів
	1	1

НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ гр. ІО-62

ДОДАТОК 2

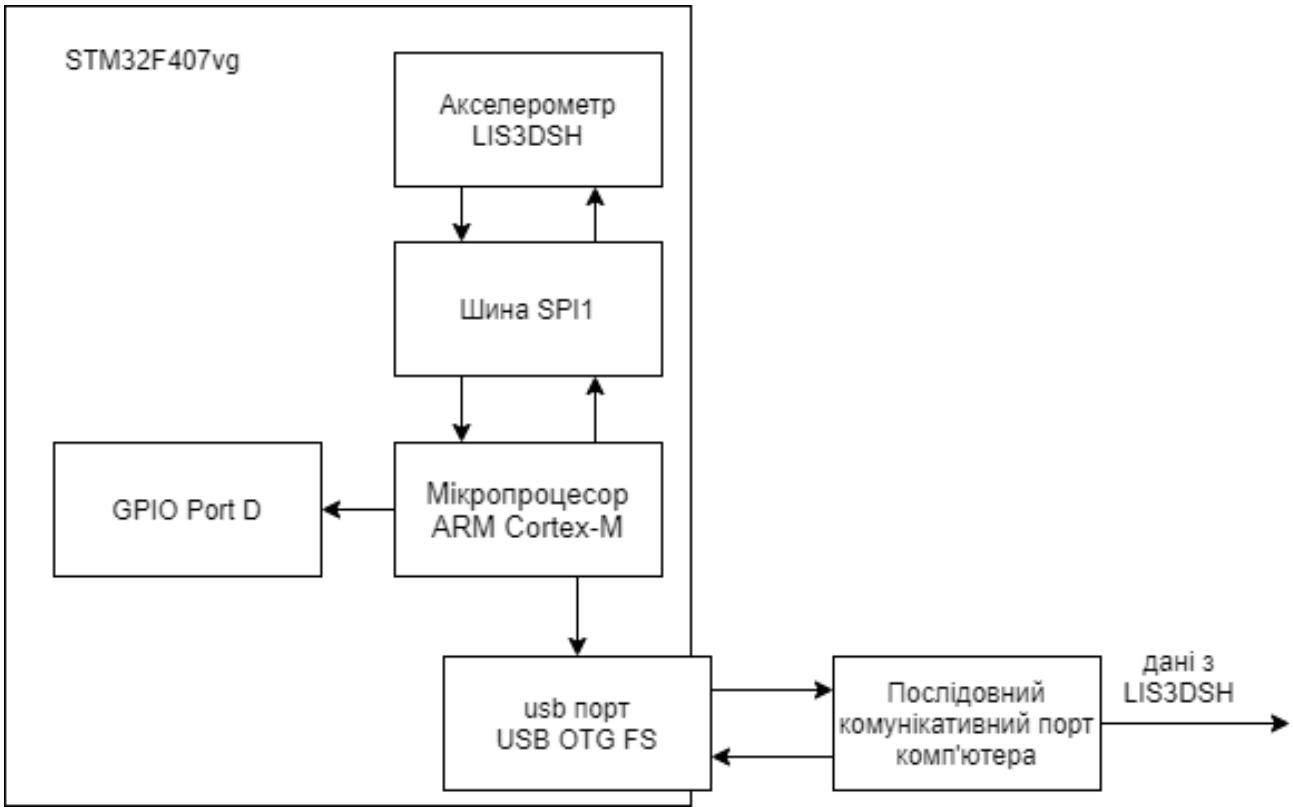
IoT система з використанням акселерометра для SmartCity

Структурна схема пристрою

ІАЛЦ.467100.005 Д2

Аркушів 1

Київ 2020р.



					ІАЛЦ.467100.005 Д2
Зм. Арк.	№ документа	Підп.	Дата		
Розробив	Копійка А.О.			IoT система з використанням акселерометра для SmartCity Структурна схема пристрою	Літ. / Аркуш 1 / Аркушів 1
Перевірив	Ткаченко В.В.				
Н.контр.	Сімоненко В.П.				НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ гр. ІО-62
Затверд.	Стіренко С. Г.				

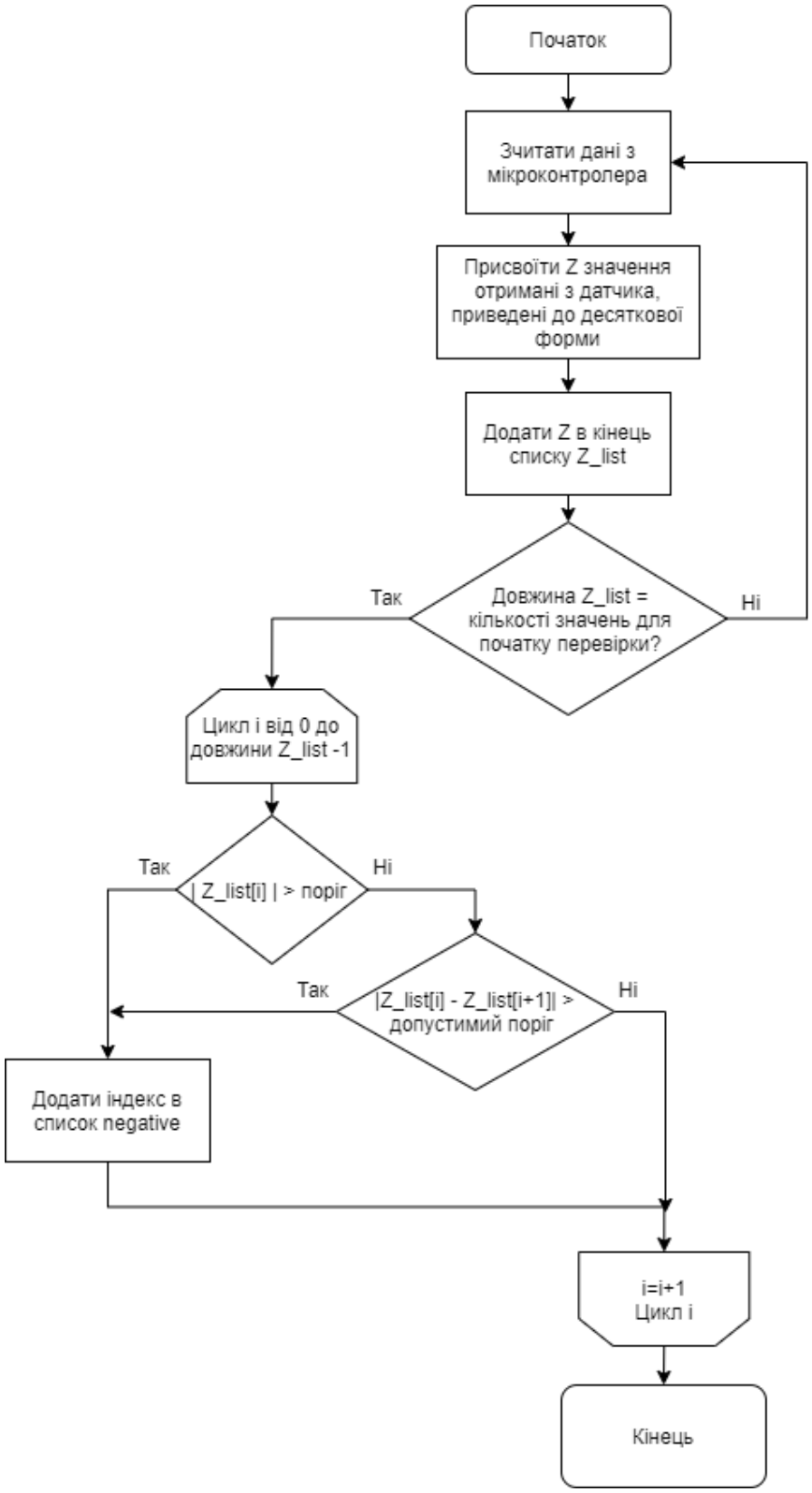
ДОДАТОК 3

IoT система з використанням акселерометра для SmartCity

Функціональна схема методу визначення ям на дорогах
ІАЛЦ.467100.006 Д3

Аркушів 1

Київ 2020 р.



Зм.		№ документа	Підп.	Дата	ІАЛЦ.467100.006 Д3		
Розробив		Копійка А.О.			IoT система з використанням акселерометра для SmartCity Функціональна схема методу визначення ям на дорогах	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Ткаченко В.В.					
Н.контр.		Сімоненко В. П.				НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ гр. ІО-62	
Затв.		Стіренко С. Г.					

ДОДАТОК 4

IoT система з використанням акселерометра для SmartCity

Текст програми

ІАЛЦ.467100.007 Д4

Аркушів 3

Київ 2020

```
#include "lis3dsh.h"
#include "usbd_cdc_if.h"
//-------------------------------------
extern SPI_HandleTypeDef hspi1;
uint8_t buf2[8]={0};
char str1[30]={0};
//-------------------------------------
static void Error (void)
{
                        LD5_ON;
}
//--------------------------------------
uint8_t SPIx_WriteRead(uint8_t Byte)
{
                        uint8_t receivedbyte = 0;
                        if(HAL_SPI_TransmitReceive(&hspi1,(uint8_t*) &Byte,(uint8_t*)
&receivedbyte,1,0x1000)!=HAL_OK)
                        {
                                Error();
                        }
                        return receivedbyte;
}
//--------------------------------------
void Accel_IO_Read(uint8_t *pBuffer, uint8_t ReadAddr, uint16_t NumByteToRead)
{
                        if(NumByteToRead>0x01)
                        {
                                ReadAddr |= (uint8_t) (READWRITE_CMD |
MULTIPLEBYTE_CMD);
                        }
                        else
                        {
                                ReadAddr |= (uint8_t)READWRITE_CMD;
                        }
                        CS_ON;
                        SPIx_WriteRead(ReadAddr);
                        while(NumByteToRead>0x00)
                        {
                                *pBuffer=SPIx_WriteRead(DUMMY_BYTE);
                                NumByteToRead--;
                                pBuffer++;
                        }
                        CS_OFF;
}
//--------------------------------------
void Accel_IO_Write(uint8_t *pBuffer, uint8_t WriteAddr, uint16_t NumByteToWrite)
{
                        CS_OFF;
                        if(NumByteToWrite>0x01)
                        {
                                WriteAddr |= (uint8_t) MULTIPLEBYTE_CMD;
                        }
                        CS_ON;
                        SPIx_WriteRead(WriteAddr);
                        while(NumByteToWrite>=0x01)
                        {
                                SPIx_WriteRead(*pBuffer);
                                NumByteToWrite--;
                                pBuffer++;
                        }
                        CS_OFF;
}
//--------------------------------------
uint8_t Accel_ReadID(void)
{
  uint8_t ctrl = 0;
                        Accel_IO_Read(&ctrl,LIS3DSH_WHO_AM_I_ADDR,1);
                        return ctrl;
}
```

```
//--------------------------------------
void Accel_AccFilterConfig(uint8_t FilterStruct)
{

}
//--------------------------------------
void AccInit(uint16_t InitStruct)
{
  uint8_t ctrl = 0;
                          ctrl=(uint8_t)(InitStruct);
                          Accel_IO_Write(&ctrl, LIS3DSH_CTRL_REG4_ADDR,1);
                          ctrl=(uint8_t)(InitStruct>>8);
                          Accel_IO_Write(&ctrl, LIS3DSH_CTRL_REG5_ADDR,1);
}
//--------------------------------------
void Accel_GetXYZ(int16_t* pData)
{
                          int8_t buffer[6];
  uint8_t ctrl,i = 0x00;
                          float sensitivity = LIS3DSH_SENSITIVITY_0_06G;
                          float valueinfloat = 0;
                          Accel_IO_Read(&ctrl, LIS3DSH_CTRL_REG5_ADDR,1);
                          Accel_IO_Read((uint8_t*)&buffer[0], LIS3DSH_OUT_X_L_ADDR,1);
                          Accel_IO_Read((uint8_t*)&buffer[1], LIS3DSH_OUT_X_H_ADDR,1);
                          Accel_IO_Read((uint8_t*)&buffer[2], LIS3DSH_OUT_Y_L_ADDR,1);
                          Accel_IO_Read((uint8_t*)&buffer[3], LIS3DSH_OUT_Y_H_ADDR,1);
                          Accel_IO_Read((uint8_t*)&buffer[4], LIS3DSH_OUT_Z_L_ADDR,1);
                          Accel_IO_Read((uint8_t*)&buffer[5], LIS3DSH_OUT_Z_H_ADDR,1);
                          switch(ctrl&LIS3DSH__FULLSCALE_SELECTION)
                          {
                                  case LIS3DSH_FULLSCALE_2:
                                          sensitivity=LIS3DSH_SENSITIVITY_0_06G;
                                          break;
                                  case LIS3DSH_FULLSCALE_4:
                                          sensitivity=LIS3DSH_SENSITIVITY_0_12G;
                                          break;
                                  case LIS3DSH_FULLSCALE_6:
                                          sensitivity=LIS3DSH_SENSITIVITY_0_18G;
                                          break;
                                  case LIS3DSH_FULLSCALE_8:
                                          sensitivity=LIS3DSH_SENSITIVITY_0_24G;
                                          break;
                                  case LIS3DSH_FULLSCALE_16:
                                          sensitivity=LIS3DSH_SENSITIVITY_0_73G;
                                          break;
                                  default:
                                          break;
                          }
                          for(i=0;i<3;i++)
                          {
                                  valueinfloat = ((buffer[2*i+1] << 8) + buffer[2*i]);
                                  //*sensitivity;
                                  pData[i]=(int16_t)valueinfloat;
                          }
}
//--------------------------------------
void Accel_ReadAcc(void)
{
                          int16_t buffer[3]={0};
                          int16_t xval, yval,zval = 0x0000;
                          Accel_GetXYZ(buffer);
                          xval=buffer[0];
                          yval=buffer[1];
                          zval=buffer[2];
                          sprintf(str1,"%06d\r\n",zval);
                          CDC_Transmit_FS((uint8_t*)str1,strlen(str1));
                          buf2[0]=0x11;
                          buf2[1]=0x55;
                          buf2[2]=(uint8_t)(xval>>8);
```

```
                        buf2[3]=(uint8_t)xval;
                        buf2[4]=(uint8_t)(yval>>8);
                        buf2[5]=(uint8_t)yval;
                        buf2[6]=(uint8_t)(zval>>8);
                        buf2[7]=(uint8_t)zval;
                        //CDC_Transmit_FS(buf2,8);

                        if((ABS(xval))>(ABS(yval)))
                        {
                                if(xval>2000)
                                {
                                        LD5_ON;
                                }
                                else if(xval<-2000)
                                {
                                        LD4_ON;
                                }
                        }
                        else
                        {
                                if(yval>2000)
                                {
                                        LD3_ON;
                                }
                                else if(yval<-2000)
                                {
                                        LD6_ON;
                                }
                        }
                        HAL_Delay(20);
                        LD3_OFF;
                        LD4_OFF;
                        LD5_OFF;
                        LD6_OFF;
}
//---------------------------------------
void Accel_Ini(void)
{
                        uint16_t ctrl = 0x0000;
                        HAL_Delay(1000);
                        if(Accel_ReadID()==0x3F) LD4_ON;
                        else Error();
                        ctrl = (uint16_t) (LIS3DSH_DATARATE_100 |
LIS3DSH_XYZ_ENABLE);
                        ctrl |= ((uint16_t) (LIS3DSH_SERIALINTERFACE_4WIRE|\

                                        LIS3DSH_SELFTEST_NORMAL|\

                                        LIS3DSH_FULLSCALE_2|\

                                        LIS3DSH_FILTER_BW_800))<<8;
                        AccInit(ctrl);
                        LD6_ON;
}
//---------------------------------------
```